Spis treści

1. Architektura systemu komputerowego
1.1. Składniki systemu komputerowego
1.2. Procesor
1.3. Magistrale systemowe
1.4. Procesor - taksonomia i układy towarzyszące
1.5. Pamięć operacyjna
1.6. Pamięć masowa
1.7. Przerwania
1.8. Operacje wejścia / wyjścia
1.9. Mechanizmy ochrony wejścia/wyjścia, pamięci, jednostki centralnej

1. Architektura systemu komputerowego

Chcielibyśmy przedstawić Wam trochę bliżej poszczególne elementy systemu komputerowego. Ta lekcja w całości będzie poświęcona sprzętowi - przedstawimy jak zbudowany jest system komputerowy, jaka jest jego logiczna struktura oraz najważniejsze elementy. Przy czym skupimy się bardziej na ich funkcjach i właściwościach, niż na budowie wewnętrznej.

1.1. Składniki systemu komputerowego

Składniki systemu komputerowego

Wstęp historyczno-teoretyczny mamy już za sobą - teraz pora zacząć mówić o konkretach ;). Zaczniemy więc od opisu strukturalnego systemu komputerowego: co wchodzi w jego skład, do czego służą poszczególne elementy i jak w dużym uproszczeniu działają. Ciągły wzrost zastosowań komputerów stymulował ich rozwój, zarówno w aspekcie funkcjonalnym, jak i technicznym. Komputer z wiekiem obrastał w piórka, aby zwiększyć swoją funkcjonalność, aby uprościć obsługę, aby połączyć się ze światem zewnętrznym, itd. ... można tu jeszcze zapewne wiele powodów podać. Dzięki temu doczekaliśmy się systemu komputerowego. Doczekaliśmy się czasów, w których komputer przekształcił się z urządzenia elektronicznego o dość jednolitej i regularnej strukturze w zespół wzajemnie powiązanych ze sobą urządzeń o rozproszonej odpowiedzialności, różnym stopniu skomplikowania i różnej szybkości działania. Podzespoły wchodzące w skład komputera są wyposażane przez producenta w oprogramowanie zapewniające jego efektywne wykorzystanie. Ten zbiór urządzeń wraz z oprogramowaniem zaczęto nazywać systemem komputerowym. Zmiana terminologiczna była również powiązana z pojawieniem się Teorii Systemów i zastosowaniem jej do opisywania współdziałania elementów komputera. Nie będę Was męczyć tymi opisami - zainteresowani mogą się w nie zagłębić samodzielnie.

Wróćmy jednak do tematu. Jak wspomniałem na samym początku tego kursu - systemem komputerowym określamy zarówno sprzęt, jak i oprogramowanie. W tej części podręcznika skupimy się na sprzęcie. Przy czym główny nacisk położymy na komputery osobiste i konstrukcje do nich zbliżone, w skrócie omawiając rozwiązania specyficzne, takie jak systemy wbudowane czy klastry obliczeniowe.

Każdy system komputerowy powinien składać się z co najmniej kilku elementów. Pamiętając o koncepcji von Neumanna, ktoś mógłby zaryzykować stwierdzenie, iż musi być ich co najmniej 4: pamięć, jednostka arytmetyczno-logiczna, jednostka sterująca oraz układ wejścia/wyjścia. W praktyce ten podział nie zawsze jest konieczny w sensie fizycznym. Istnieją bowiem tak zwane komputery jednoukładowe (mikrokontrolery - takie informatyczne *wash-and-go*, czyli wszystko w jednym). Jednak nawet w tym przypadku wyraźnie można wydzielić odpowiednie części funkcjonalne w układzie scalonym. Dlatego zaryzykujemy stwierdzenie, iż w modelu, którym posługujemy się w tym kursie, każdy współczesny system komputerowy składa się z co najmniej jednej **jednostki centralnej**, która łączy następujące elementy występujące w koncepcji von Neumanna: jednostkę arytmetyczno-logiczną (która liczy) oraz jednostkę sterującą (której rolą jest interpretacja programu – jak liczyć). Taką jednostkę centralną będziemy nazywali **procesorem.** Oprócz procesora system komputerowy musi mieć co najmniej jednego rodzaju **pamięć** (przechowującą dane wejściowe do obliczeń wraz z ich wynikami) oraz **układy wejścia-wyjścia** (pozwalające na komunikację ze światem zewnętrznym - wprowadzenie programu, danych do obliczeń czy też prezentację wyników).

Jeśli ograniczymy się do klasycznych komputerów ogólnego stosowania, możemy ten podział jeszcze bardziej uszczegółowić: system komputerowy ogólnego stosowania składa się z jednej bądź kilku **jednostek centralnych** oraz kilku **sprzętowych kontrolerów urządzeń** (ang. *device controller*) połączonych wspólną **magistralą danych** umożliwiającą współpracę ze wspólną **pamięcią** (patrz rysunek poniżej).


[Architektura systemu komputerowego]
Architektura systemu komputerowego

Każde urządzenie wchodzące w skład systemu komputerowego ma zdefiniowany standard połączenia się z jednostką centralną oraz posiada swój **kontroler** (sprzętowy – bez względu na to, czy jest to dysk twardy, karta grafiki, czy drukarka). Kontroler ma za zadanie zapewnić uporządkowany, kontrolowany dostęp do wspólnej pamięci lub innych zasobów, pomimo współbieżnego działania z jednostką centralną. Z drugiej strony – kontroler (wraz z odpowiednim sterownikiem programowym, o którym powiemy przy omawianiu systemów operacyjnych) zapewnia zunifikowany dostęp do urządzeń pełniących tę samą rolę, lecz wyprodukowanych przez różnych producentów.

Ostatecznie możemy powiedzieć, że na architekturę komputera składają się:

1. **Jednostka centralna (CPU)**, czyli przynajmniej jeden procesor. Przynajmniej - gdyż coraz częściej wykorzystywane są dwa (lub więcej) procesory (nawet jeśli są w postaci jednego układu scalonego).
2. **Magistrala systemowa** poprzez którą komunikują się i przesyłają dane komponenty systemu komputerowego.
3. **Wspólna pamięć**, która służy do przechowywania danych i kodu instrukcji procesora, ale - zgodnie z modelem von Neumanna - ma jednolitą strukturę.
4. **Urządzenia wejścia-wyjścia** (np. klawiatura - monitor), dzięki którym możliwa jest komunikacja ludzi z komputerem.

W następnych segmentach tej lekcji postaramy się wyjaśnić dokładniej budowę oraz działanie wyżej wymienionych elementów systemu komputerowego.

1.2. Procesor

Co to jest?

Czasem jest utożsamiany z całym systemem komputerowym, czasem sądzi się, że to on i tylko on decyduje o wydajności systemu, a już przekonanie, iż w procesorze kryje się inteligencja komputera - jest powszechne. Niestety, żadne z powyższych stwierdzeń nie jest prawdą. Procesor nie stanowi systemu komputerowego w całości, bez pozostałych komponentów nie jest do niczego przydatny, co więcej - koszt procesora wcale nie musi być najistotniejszą pozycją w koszcie systemu komputerowego. Wpływ procesora na wydajność jest olbrzymi - to fakt. Lecz nawet najszybszy procesor, który współpracuje ze zbyt małą ilością pamięci, bądź też z pamięcią zbyt wolną, nigdy nie będzie w pełni wykorzystany. Podobnie w przypadku innych, nieprawidłowo dobranych komponentów systemu komputerowego. A co do inteligencji ... no cóż, warto wyraźnie podkreślić, że współczesne komputery z inteligencją sensu stricte nie mają nic wspólnego. A skoro tak - to i w procesorze inteligencji jest tyle, co w kalkulatorze czy liczydłach ...

Czym zatem jest procesor dla systemu komputerowego? W pierwszym przybliżeniu możecie go potraktować jako bardzo szybki i trochę bardziej skomplikowany kalkulator. Procesor potrafi przekształcać liczby binarne (dodawać, odejmować, mnożyć, dzielić, negować, itp.) w sposób definiowany poprzez program składający się również z liczb binarnych. A bardziej formalnie:

> **Definicja**
>
> procesor (spotyka się również określenie CPU, ang. *Central Processing Unit*) jest tym komponentem w systemie komputerowym, który interpretuje program i zgodnie z nim przetwarza dane.

Ze względu na wspomnianą "programowalność" procesora, to właśnie jego istnienie determinowało zaliczenie pierwszych maszyn liczących do komputerów. Procesor wykonany w postaci jednego układu scalonego zwyczajowo nazywa się mikroprocesorem.

Podstawową funkcją większości procesorów (niezależnie od ich konstrukcji fizycznej) jest wykonanie programu, czyli sekwencji instrukcji przechowywanych w pamięci. Wykonanie każdej z tych instrukcji jest również rozbite na kroki. Zazwyczaj wyróżnia się następujące główne kroki potrzebne do wykonania pojedynczej instrukcji:

- *pobieranie rozkazów* - czyli odczyt poleceń z pamięci. Każde polecenie jest liczbą (możecie je traktować jako numer rozkazu). Ile poleceń "rozumie" nasz procesor - to już kwestia konstrukcji, ale zawsze jest to skończona liczba. Położenie instrukcji w pamięci jest zdefiniowane przez specjalny blok, nazywany licznikiem rozkazów (ang. program counter, PC), który przechowuje adres aktualnie wykonywanej instrukcji w pamięci operacyjnej. Informacje o bieżącym stanie wykonywania programu (takie jak wyniki porównań czy bity znaczników) są natomiast przechowywane zazwyczaj w osobnym rejestrze, nazywanym słowem stanu programu (ang. program status word, PSW). Po pobraniu następuje zwiększenie PC o długość pobranego rozkazu (w jednostkach pamięci, czyli najczęściej w bajtach). Natomiast w przypadku konieczności wykonania skoku - zawartość PC może być zmieniana przez odpowiednią instrukcję procesora.
- *interpretacja rozkazów* - po odczytaniu polecenia z pamięci, w pierwszym etapie przetwarzania, procesor interpretuje polecenie. W trakcie tego kroku instrukcja jest rozbijana na części, które mają znaczenie dla poszczególnych bloków procesora. Sposób w jaki procesor interpretuje instrukcje, jest zależny od jego architektury i zestawu instrukcji (instruction set architecture, ISA). Przykładowo, w jednej długiej liczbie będącej instrukcją, początkowa część bitów (nazywana opcode) jest znacznikiem operacji do wykonania. Pozostała część bitów zazwyczaj zawiera informacje wymagane przez instrukcję, taką jak adresy składników dla operacji dodawania.
  Przykład dekodowania instrukcji dodawania dla procesora MIPS32:

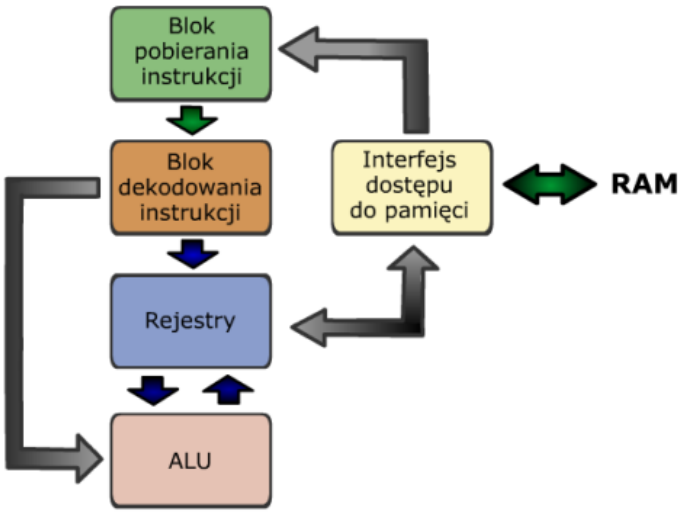
  MIPS 32 – Instrukcja dodawania

001000	00001	00010	0000000101011110
OP Code	Addr 1	Addr 2	**Wartość**

- *wykonywanie rozkazów* - po pobraniu i zdekodowaniu przychodzi pora na rzeczywiste wykonanie polecenia. Podczas tego kroku różne moduły CPU współpracują ze sobą w celu wykonania polecenia. Rozpatrzmy jako przykład dodawanie dwóch liczb: jednostka arytmetyczno-logiczna (ALU) zawiera układ elektroniczny, który jest w stanie wykonać proste operacje arytmetyczno - logiczne (jak dodawanie, czy operacje logiczne na bitach) dla danych wejściowych znajdujących się w rejestrach. Układ ten jest więc uruchamiany, a następnie wynik dodawania jest umieszczany w rejestrze wyjściowym. Jeśli w wyniku dodawania powstanie zbyt duża liczba (taka, której nie można zapamiętać), znacznik przepełnienia w rejestrze zawierającym wynik może również być ustawiony.
  Zwróćcie uwagę na fakt, iż wykonanie instrukcji odbywa się na danych znajdujących się w rejestrach. Z punktu widzenia procesora pobranie danych z pamięci operacyjnej do rejestru, lub też odesłanie wyników do pamięci operacyjnej, jest niezależną instrukcją.
- *zapis wyników* - finalna faza - efekty pracy procesora są zapamiętywane w rejestrze wyjściowym, skąd mogą zostać odesłane do pamięci operacyjnej, lub też mogą być wykorzystane w kolejnym kroku obliczeń (przy wykonywaniu kolejnej instrukcji). Część instruckji zmienia zawartość licznika instrukcji (PC) w celu wykonania skoku w programie (dzięki tej funkcji możliwe jest tworzenie pętli i instrukcji warunkowych). Inne natomiast zmieniają ustawienie znaczników w specjalnym rejestrze (zwanym znacznikowym). Mogą być one później wykorzystywane do podejmowania decyzji o skoku, itp ... ale w tym wypadku zaczynamy się już zbliżać do poziomu programowania w

asemblerze, co nie jest tematyką podręcznika. Wystarczy abyście zapamiętali, że wykonanie kolejnych instrukcji kończy się zapisaniem wyników, lecz wynik niekoniecznie musi być zapisywany w pamięci operacyjnej 😊 .

W nowoczesnych procesorach wymienione etapy cyklu rozkazu są zazwyczaj zorganizowane w potok (ang. *pipeline*): w każdym takcie zegara różne fazy (pobieranie, dekodowanie, wykonanie, zapis) mogą być wykonywane równolegle dla kilku różnych instrukcji. Wiele współczesnych procesorów stosuje także wykonanie poza kolejnością (ang. *out-of-order execution*), aby lepiej wykorzystać jednostki wykonawcze. Z punktu widzenia programisty efekty są jednak takie, jak gdyby instrukcje wykonywały się kolejno, w podanej w programie kolejności (por. Instruction pipelining, Out-of-order execution).



Działanie procesora

Aby procesor miał możliwość wykonywania powyższych zadań musi dysponować małą pamięcią wewnętrzną, która wymagana jest do czasowego przechowywania danych i rozkazów (procesor musi np. pamiętać lokalizację poprzedniego rozkazu po to, by "odnaleźć" rozkaz następny). Większość (choć nie wszystkie) nowoczesnych systemów komputerowych opiera się na założeniu, że procesor nie operuje na danych bezpośrednio umieszczonych w pamięci operacyjnej, lecz za każdym razem następuje przekazanie zarówno danych, jak i wyników obliczeń z / do pamięci za pośrednictwem rejestrów. Taki model jest typowy dla tzw. architektur typu *load–store* (RISC), takich jak ARM, RISC-V czy MIPS, w których tylko specjalne instrukcje ładowania i zapisu mają dostęp do pamięci, a wszystkie operacje arytmetyczne działają wyłącznie na danych w rejestrach. W architekturach typu x86 część instrukcji może formalnie operować bezpośrednio na komórkach pamięci, jednak wewnętrznie są one zwykle rozbijane na prostsze mikrooperacje wykonywane na rejestrach (por. Load–store architecture, ARM architecture family, RISC-V, x86 instruction set).

Rejestry

> ### Definicja
>
> Rejestrami procesora będziemy nazywali wydzielone, bardzo szybkie komórki pamięci, umieszczone w samym procesorze. Umożliwiają one szybki dostęp do potrzebnych danych lub rozkazów.

Zatem, jak to działa? Ponieważ procesor to nic innego jak taki duży i skomplikowany kalkulator, który dodatkowo pobiera informację o czynności do wykonania z pamięci operacyjnej, zobaczcie jak przebiegałoby policzenie przykładowego wyrażenia na kalkulatorze. Przyjmijmy, że komputer ma za zadanie obliczyć wartość następującego wyrażenia:

(x+y)*(x-z)

Jeśli liczymy to my, korzystając z kalkulatora, to możemy uzyskać wynik wykonując następujące czynności:

1. Najpierw ustalam, jaka jest wartość danych oznaczonych we wzorze literami x, y, z.
2. Wpisuję wartość x
3. Naciskam przycisk +
4. Wpisuję wartość y
5. Naciskam przycisk =
6. Odczytuję obliczoną wartość z wyświetlacza i zapisuję ją na kartce oznaczając jako t
7. Naciskam przycisk AC (kasuj)
8. Wpisuję wartość x
9. Naciskam przycisk -
10. Wpisuję wartość z
11. Naciskam przycisk =
12. Naciskam przycisk *

13. Wpisuję wartość t
14. Naciskam przycisk =
15. Odczytuję wynik działania z wyświetlacza i zapisuję jako obliczoną wartość w.

Procesor realizuje to zadanie w zbliżony sposób, przy czym każda podstawowa czynność to pojedynczy rozkaz procesora. Załóżmy, że rozkazy procesora mogą mieć jedną z trzech postaci (zbliżamy się w ten sposób do maszyny Turinga ;)):

- pobierz dane z pamięci operacyjnej do rejestru (rejestr to "prywatna" jednostka pamięci należąca do procesora);
- wykonaj operacje arytmetyczne na danych w rejestrach;
- prześlij dane z rejestru do pamięci operacyjnej.

Załóżmy też, że przed uruchomieniem obliczeń, gdzieś w pamięci operacyjnej, pod adresami nazwanymi dla wygody x, y, z (zamiast np. 0xfe035ad8, bo tak naprawdę tylko takie adresy akceptuje procesor) umieszczone są wartości danych, dla których ma zostać wykonane działanie (x+y)*(x-z). Lista rozkazów dla procesora realizujących obliczenie wyrażenia (bez pobierania z zewnątrz danych wejściowych i wyświetlania wyników) będzie więc miała postać:

1. Pobierz wartość spod adresu x do rejestru R1
2. Pobierz wartość spod adresu y do rejestru R2
3. Dodaj zawartość rejestrów R1 i R2 i wynik wstaw do R1
4. Prześlij wynik z rejestru R1 pod adres t
5. Pobierz wartość spod adresu x do rejestru R1
6. Pobierz wartość spod adresu z do rejestru R2
7. Odejmij zawartość rejestru R2 od rejestru R1 i wynik wstaw do R1
8. Pobierz wartość spod adresu t do rejestru R2
9. Pomnóż zawartość rejestrów R1 i R2 i wynik wstaw do R1
10. Prześlij zawartość z rejestru R1 pod adres w.

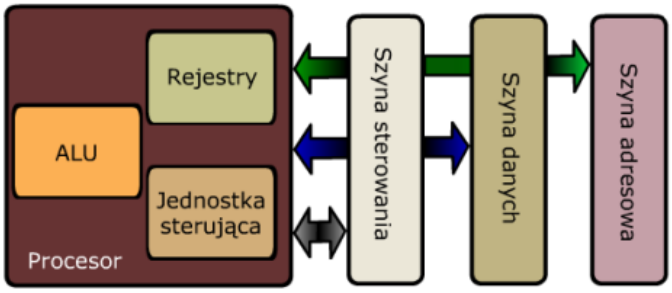
I w zasadzie to wszystko co potrafi zrobić procesor ... ;). Tyle, że te działania wykonuje bardzo szybko.
Liczba rejestrów we współczesnych procesorach różni się w zależności od producenta. Kiedyś można było stwierdzić, że im więcej rejestrów, tym komputer szybszy. Aktualnie to stwierdzenie niekoniecznie musi być prawdziwe, lecz zazwyczaj odpowiada prawdzie. Poniżej znajdziecie liczbę rejestrów wykorzystywanych w wybranych historycznych i współczesnych procesorach:

Procesor	**Liczba rejestrów całkowitoliczbowych**	**Liczba rejestrów zmiennoprzecinkowych**
Zgodne z x86-64 (np. Intel Core)	16	16
Pentium 4	8	8
Athlon MP	8	8
Opteron 240	16	16
Itanium 2	128	128
UltraSPARC IIIi	32	32
PowerPC / Power ISA	32	32

Powyższe przykłady mają charakter historyczny i pokazują zróżnicowanie liczby rejestrów w procesorach projektowanych głównie na przełomie lat 90. i 2000. Współczesne popularne architektury RISC wyglądają bardzo podobnie: przykładowo w trybie AArch64 (ARMv8-A / ARMv9-A) dostępnych jest 31 64-bitowych rejestrów ogólnego przeznaczenia oraz 32 128-bitowe rejestry wektorowe/zmiennoprzecinkowe (v0–v31), natomiast standardowa konfiguracja architektury RISC-V udostępnia 32 rejestry całkowitoliczbowe i – przy włączonym rozszerzeniu zmiennoprzecinkowym – dodatkowe 32 rejestry zmiennoprzecinkowe (por. dokumentacje AArch64 i RISC-V: ARM Architecture Reference Manual for A-profile, RISC-V Specifications).

Budowa wewnętrzna procesora

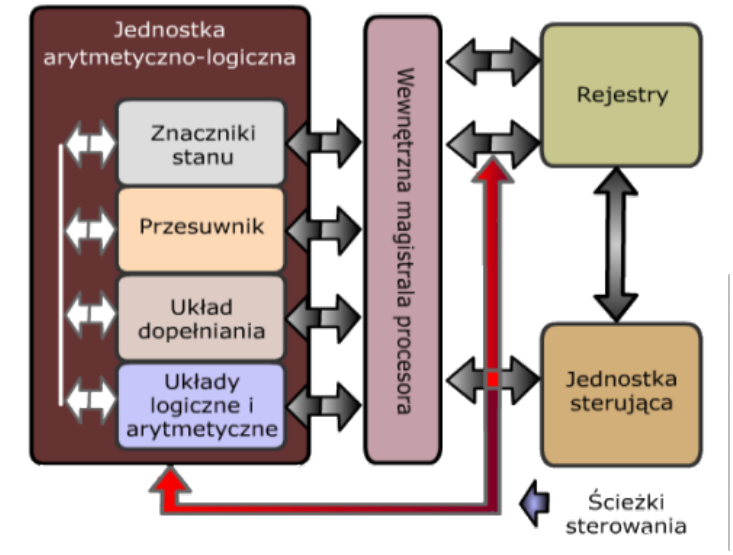
Każdy współczesny procesor jest zbudowany z kilku zespołów funkcjonalnych. Zaczniemy od najbardziej uproszczonego opisu. Poniżej przedstawione są dwa rysunki, które ukazują nam budowę procesora w pierwszym przybliżeniu. Pierwszy z nich przedstawia główne bloki funkcjonalne procesora, w zasadzie odpowiadające odpowiednim blokom funkcjonalnym ze schematu działania procesora. Blok pobierania instrukcji i blok dekodowania instrukcji są umieszczone w jednostce sterującej, natomiast jawnie pojawiają się rejestry i jednostka arytmetyczno-logiczna. Dokładniej rozrysowana jest tutaj komunikacja procesora z pamięcią, ale o tym będzie więcej w następnym segmencie.



Budowa procesora

Komunikacja między procesorem a resztą systemu komputerowego odbywa się poprzez magistralę systemową, której poszczególne składniki (szyny danych, adresowa i sterowania) mogą, ale nie muszą być fizycznie od siebie odseparowane.

Nieco dokładniej strukturę wewnętrzną procesora przedstawia kolejny rysunek. Element określony jako **wewnętrzna magistrala procesora** zajmuje się przesyłaniem danych między różnymi rejestrami a ALU (w rzeczywistości ALU operuje jedynie na danych zaczerpniętych z wewnętrznej pamięci procesora). Rysunek przedstawia również w zarysie poszczególne elementy jednostki arytmetyczno-logicznej.



Procesor: struktura wewnętrzna

Zapamiętajcie, że tak jak procesor jest "sercem" komputera, tak ALU jest "sercem" procesora. Rejestry są używane do przechowywania danych wewnętrznych w procesorze, a niektóre z nich zawierają informacje potrzebne do zarządzania porządkowaniem rozkazów (np. słowo stanu programu), pozostałe zawierają dane przeznaczone dla ALU, pamięci i modułów wejścia-wyjścia lub dane przekazane przez te jednostki. Wewnętrzne ścieżki danych są używane do przenoszenia danych pomiędzy rejestrami oraz między rejestrami a ALU. Zewnętrzne ścieżki danych łączą rejestry z pamięcią i modułami wejścia-wyjścia, często za pomocą magistrali systemowej. Jednostka sterująca koordynuje wykonanie operacji wewnątrz procesora.

1.3. Magistrale systemowe

System komputerowy zawiera pewną liczbę magistrali. Najważniejsza z nich to magistrala systemowa (niektóre systemy wykorzystują kilka magistrali systemowych) łącząca najważniejsze podzespoły komputera, czyli procesor, pamięć i układy wejścia-wyjścia.

> **Definicja**
>
> Magistrala to zespół linii oraz układów przełączających służących do przesyłania sygnałów między połączonymi urządzeniami. W komputerach jest rodzajem "autostrady", którą dane przenoszą się pomiędzy poszczególnymi elementami komputera.

Zazwyczaj magistrala przesyła całe słowo maszynowe wykorzystywane przez wiele urządzeń. Jedno słowo maszynowe to 8, 16, 32, 64 lub więcej bitów, czyli napięć elektrycznych. Tak więc jedno słowo można przesłać albo jednym "przewodem" (połączeniem elektrycznym) jeden za drugim, albo jednocześnie 8, 16, 32, 64 połączeniami (w zależności od długości słowa). Pierwsze rozwiązanie nazywamy szeregowym, drugie - równoległym.

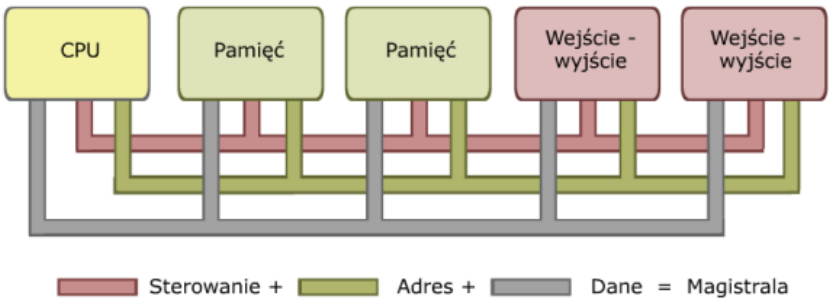
Historycznie magistrala systemowa była w większości przypadków magistralą równoległą. We współczesnych komputerach osobistych i serwerach coraz częściej stosuje się szybkie szeregowe interfejsy punkt-punkt (np. PCI Express), jednak w celach dydaktycznych nadal stosuje się model wspólnej magistrali systemowej.

Struktura magistrali

Magistrala systemowa składa się z wielu oddzielnych linii o określonym znaczeniu lub funkcji. Linie zawarte w magistrali można podzielić na trzy grupy (patrz rysunek niżej):

- linie danych;
- linie adresów;
- linie sterowania;

Opcjonalnie mogą pojawić się również linie zasilania, jednak nie zawsze one występują.



Połączenie magistrali z elementami systemu komputerowego

Linie danych służą do przenoszenia danych pomiędzy komponentami systemu. Linie te zwane są **szyną danych** (ang. *data bus*). Liczba linii szyny określa jej *szerokość* (przeważnie występuje 8, 16, 32, 64 linii), czyli ilość bitów, które można jednocześnie przesyłać. Szerokość szyny danych jest jednym z głównych czynników określających wydajność systemu komputerowego. Przykładowo, jeśli szyna danych ma szerokość 16 bitów, a wykonywany rozkaz 32 bity, to procesor chcąc nie chcąc musi łączyć się z modułem pamięci dwukrotnie w czasie tego cyklu rozkazu. Z tego względu optymalnym rozwiązaniem jest stosowanie szyny danych dopasowanej do szerokości rejestrów danych w procesorze.

Linie adresowe wykorzystywane są do określania skąd i dokąd przepływają informacje przesyłane szyną danych. Na przykład procesor, który chce odczytać dane z pamięci, umieszcza adres tych danych na szynie adresowej, a chcąc zapisać efekt ich wykonania wysyła na szynę adresową adres pamięci (lub modułu we-wy), do którego chce przesłać tenże wynik.
Istotnym parametrem tej szyny jest jej *szerokość*. Szerokość szyny adresowej określa maksymalną pojemność pamięci systemu, a ściślej rzecz ujmując przestrzeń adresową, która nie musi być pamięcią, ale np. urządzeniem we-wy. Zwykle bity mniej znaczące służą do adresowania pamięci lub określonych portów wewnątrz modułu we-wy, zaś bity bardziej znaczące do wybrania określonego modułu.

Alternatywnie istnieją układy, w których szyna adresowa i szyna danych są multipleksowane — dane i adresy przesyłane są tą samą linią, a ich rozdzieleniem zajmuje się układ multipleksera. Rozwiązanie to spotyka się częściej w prostszych układach lub mikrokontrolerach z ograniczoną liczbą wyprowadzeń niż w rozbudowanych komputerach osobistych.

Linie sterowania (nazywane czasem również kontrolnymi) zapewniają regulację dostępu do linii danych i linii adresowych oraz umożliwiają sterowanie ich wykorzystaniem. Sygnały sterujące przesyłane pomiędzy modułami systemu komputerowego zawierają:

- rozkazy, które określają jakie operacje mają być wykonane;
- informacje regulujące czas (taktujące), określające ważność danych i adresów.

Istnieją dwa główne typy protokołu pracy magistrali:

- synchroniczny – czas przesyłania danych kontrolowany jest przez CPU,
- asynchroniczny – wolniejsza pamięć lub układ I/O może sygnalizować potrzebę spowolnienia transmisji.

Reasumując, działanie magistrali systemowej opiera się na następujących zasadach:

- jeśli pewien moduł (X) zamierza przesłać dane do modułu Y, to musi:
    1. uzyskać dostęp do magistrali;
    2. przekazać dane za pośrednictwem magistrali;
- jeśli moduł Y chce pozyskać dane od X, musi:
    1. uzyskać dostęp do magistrali;
    2. przekazać zapotrzebowanie do modułu X poprzez odpowiednie linie sterowania i adresowe.

Przykład przebiegu operacji zapisu do pamięci (synchronicznie):

1. CPU ustala adres i „kładzie” go na magistrali (szyna adresowa)
2. CPU aktywuje CS (Chip Select)
3. Pamięć pobiera adres
4. CPU „kładzie” dane na magistrali (szyna danych)
5. CPU aktywuje funkcję „write” (szyna sterowania)
6. Pamięć pobiera dane i zapisuje je pod wcześniej pobranym adresem
7. CPU deaktywuje funkcję „write”
8. CPU deaktywuje szynę danych

Przykład przebiegu operacji odczytu (synchronicznie):

1. CPU ustala adres i „kładzie” go na magistrali;
2. CPU aktywuje CS (Chip Select);
3. Pamięć pobiera adres
4. CPU aktywuje funkcję „read”
5. Pamięć „kładzie” dane na magistrali
6. CPU pobiera dane
7. CPU deaktywuje funkcję „read”
8. CPU deaktywuje szynę danych

1.4. Procesor - taksonomia i układy towarzyszące

Pojęcia podstawowe charakteryzujące procesor

Często spotkacie się z różnymi określeniami pojawiającymi się w opisie procesora, o których jeszcze nie mówiliśmy. Ten rozdział ma za zadanie przybliżyć Wam, co oznaczają pojęcia słowa maszynowego, architektury procesora, itp... Najbardziej ogólnym jest

> **Definicja**
>
> architektura procesora, która w uproszczeniu oznacza wszystkie najważniejsze z punktu widzenia budowy i funkcjonalności cechy układu, które związane są zarówno z jego modelem programowym, jak i fizyczną budową.

> **Definicja**
>
> Model programowy procesora (ang. *Instruction Set Architecture*) to zestaw instrukcji procesora oraz innych jego cech (np. liczby i numeracji rejestrów czy sposobu adresowania pamięci) istotnych z punktu widzenia programisty, które są niezależne od ich wewnętrznej realizacji w procesorze.

Z kolei fizyczna budowa układu nazywana jest często mikroarchitekturą (ang. *microarchitecture*). To szczegółowa implementacja danego modelu programowego, która związana jest z rzeczywistym wykonywaniem operacji przez procesor. Co ciekawe, operacje te mogą być realizowane w zupełnie odmienny sposób niż wynikałoby to z modelu programowego. Przykładem może być tutaj wykonywanie kilku instrukcji poza kolejnością lub przewidywanie skoków warunkowych, czego nie znajdziemy w modelu programowym.

Mikroarchitekturę procesora przedstawia się jako zbiór diagramów określających poszczególne jego elementy, takie jak bloki funkcjonalne czy jednostki wykonawcze oraz występujące między nimi połączenia. W mikroarchitekturze nie określa się natomiast faktycznej, fizycznej implementacji obwodów logicznych w krzemowej strukturze układu, a więc składających się na nie bramek i wchodzących w ich skład tranzystorów. W tym miejscu warto podkreślić, że procesory realizujące ten sam model programowy mogą w znaczący sposób różnić się od siebie swoją mikroarchitekturą – dobrym przykładem są układy firm AMD i Intel, które są zgodne z programową architekturą x86, ale są całkowicie różnymi układami pod względem swojej mikroarchitektury.

W tym rozdziale zajmiemy się głównie mikroarchitekturą procesorów i różnymi określeniami z nią związanymi.

Słowo maszynowe, szerokość szyny danych

Sposób w jaki procesor pamięta przetwarzane liczby całkowite jest jego cechą konstrukcyjną, która wpływa na praktycznie wszystkie dostępne funkcje. Mówiliśmy wcześniej o rejestrach. Pojedynczy rejestr jest określoną liczbą bitów, czyli logicznych zer lub jedynek. W zależności od ilości bitów przechowywanych w rejestrze, i co za tym idzie - przetwarzanych przez procesor, zmienia się rozmiar i precyzja danych na których operuje. Liczba bitów przetwarzanych w jednym cyklu procesora przez ALU jest nazywana jego **słowem maszynowym** (ang. *word size*). W praktyce słowo maszynowe jest często powiązane z **szerokością szyny danych** (ang. *data path width*) procesora, ale pojęcia te nie muszą być zawsze tożsame. W opisach spotkacie również określenia takie jak *bit width* czy *integer precision*, odnoszące się do liczby bitów używanych do reprezentacji liczb całkowitych.

Napotykając określenie - procesor 32 bitowy, 64 bitowy, itd. powinniście wiedzieć, że chodzi tu o długość rejestrów danych, a co za tym idzie - długość słowa przetwarzanego przez ALU procesora w jednej instrukcji.

Długość poszczególnych rejestrów różni się dla różnych procesorów i architektur, czasami nawet różni się dla poszczególnych modułów jednego procesora. Przykładowo 8-bitowy procesor jest w stanie w jednej instrukcji przetwarzać 8 bitów, co oznacza, że może pracować z liczbami od 0 do 255. Lecz jeśli rejestr używany do przekazywania adresów (szerokość szyny adresowej) miałby również 8 bitów, to komputer oparty na takim procesorze miałby fizycznie ograniczoną wielkość pamięci do 256 komórek (maksymalna możliwa pojemność RAM wynosiłaby 256 bajtów!). Dlatego też zazwyczaj szyna i rejestry adresowe są dłuższe, dla procesorów 8-bitowych miały zwyczajowo 16 bitów, co dawało 65536 możliwych adresów i 64 kB RAM. Tyle właśnie maksymalnie mogły mieć komputery ZX Spectrum, Atari czy Commodore. Dla CPU wykorzystującego 32 bity do reprezentowania adresu, maksymalna możliwa pamięć (czyli **przestrzeń adresowa**) wynosi 4 GB (giga bajty). Oczywiście, istnieją metody wirtualnego powiększania pamięci, choćby poprzez stronicowanie, które pozwalają zwiększyć przestrzeń adresową bez poszerzania szyny danych, lecz ich stosowanie zawsze prowadzi do dodatkowego narzutu.

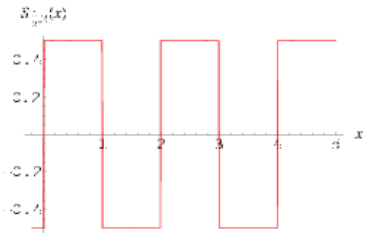
Dłuższe słowo maszynowe czy szersza szyna adresowa wymagają większej ilości tranzystorów i zbudowanych z nich bramek logicznych do obsługi, co pociąga za sobą większy koszt, stopień skomplikowania czy też zużycie energii. Dlatego też często stosuje się 4- lub 8-bitowe mikrokontrolery w systemach osadzonych, mimo że od dawna dostępne są układy 16, 32, 64, czy nawet 128-bitowe. Jednak w systemach

komputerowych stosowanych jako komputery osobiste czy serwery, zysk ze wzrostu szybkości czy też dostępnej pamięci dla procesorów o większym słowie maszynowym, zdecydowanie przewyższa zwiększony koszt czy zużycie energii przez taki procesor.

Zatem, wybierając procesor do komputera osobistego warto kupować taki, który ma jak najdłuższe słowo maszynowe, czyli współcześnie - procesory 64 bitowe. Mimo, że ciągle większość oprogramowania nie potrafi wykorzystać tej cechy, to jednak procesor dłużej pozostanie nowoczesny i wydajny.

Częstotliwość zegara

Większość procesorów, podobnie jak większość sekwencyjnych układów elektronicznych, jest z natury synchroniczna. To oznacza, że aby mogły wykonywać swoje zadania muszą mieć podany sygnał synchronizujący. Taki sygnał, nazywany zegarowym, najczęściej przyjmuje formę prostokątnej fali w czasie:



Patrząc na powyższy rysunek możemy powiedzieć, że jest to po prostu taki sygnał, co pojawia się i znika ... jest napięcie - nie ma, jest - nie ma, jest - nie ma ... i tak miliony razy na sekundę ;). Miarą tego, ile razy sygnał zmienia się w ciągu sekundy, jest częstotliwość. Ale co to ma wspólnego z procesorem?

Nawet nie wyobrażacie sobie jak duzo. Powiedzieliśmy wcześniej, że procesor jest urządzeniem synchronicznym, co oznacza, że kolejne polecenie (kolejny krok wykonania instrukcji) jest inicjowane zmianą sygnału zegarowego. Co oznacza, iż im większa częstotliwość (ilość zmian sygnału zegarowego) - tym szybciej nasz procesor jest w stanie liczyć. Niestety, każdy układ elektroniczny ma swoją graniczną częstotliwość powyżej której nie będzie działał. Obliczając maksymalny czas propagacji sygnału elektrycznego przez obwody procesora (zmiany napięcia poruszają się z prędkością bliską prędkości światła) jesteśmy w stanie wyznaczyć graniczną częstotliwość zegara. W praktyce, maksymalna możliwa częstotliwość jest zawsze niższa od granicznej. Jest to powodowane poprzez niedokładności wykonania procesora, zbyt dużą ilość wydzielanego ciepła, problemy z utrzymaniem całego układu w synchronicznej pracy, etc. Dlatego producenci wyznaczają doświadczalnie maksymalną dopuszczalną częstotliwość zegara dla każdego egzemplarza procesora.

Wiemy już, że częstotliwość zegara jest ważnym elementem wydajności procesora. Jednakże nie znaczy to bezpośrednio, że procesor taktowany wyższą częstotliwością będzie zawsze szybszy niż inny - taktowany niższą. Taki związek nie zachodzi między innymi dla procesorów o różnej długości słowa maszynowego - 32 bitowy procesor taktowany zegarem 1 GHz będzie prawdopodobnie wolniejszy niż 64 bitowy taktowany 800 MHz w typowych zastosowaniach. Wydajność procesora jest w dużym stopniu uwarunkowana również jego budową wewnętrzną (mikroarchitekturą), dlatego procesory o tej samej częstotliwości mogą osiągać bardzo różną rzeczywistą wydajność w zależności od rodzaju obciążenia i wykorzystywanych instrukcji.

Lista instrukcji procesora

Zapewne każdy z Was wie, że wymiana procesora na szybszy nie pociąga za sobą wymiany wszystkich programów. Wiecie również, że program pisany dla Mac'a nie da się uruchomić na PC. Dlaczego?

Poszczególne procesory mogą mieć różne listy instrukcji, czyli rozkazów które rozumieją i potrafią wykonać. Zazwyczaj procesory jednej rodziny mają taką samą listę instrukcji (np. x86 to wspólna nazwa dla zestawu instrukcji rozumianych przez wszystkie procesory Intela od I386 do współczesnych rozwiązań oraz procesory AMD, ale totalnie nieznana procesorom serii UltraSPARC czy ARM).

Przy projektowaniu zestawów instrukcji dla nowej rodziny procesorów zazwyczaj przyjmuje się jedno z dwóch podejść:

- **CISC** (*Complex Instruction Set Computers*) – tworzona jest lista rozkazów o następujących cechach:
    - duża liczba instrukcji;
    - mała optymalizacja – niektóre rozkazy potrzebują dużej liczby cykli procesora do wykonania;
    - występowanie złożonych, specjalistycznych rozkazów;
    - duża liczba trybów adresowania;
    - do pamięci może się odwoływać bezpośrednio duża liczba rozkazów;
    - mniejsza od RISC-ów częstotliwość taktowania procesora;
    - powolne działanie dekodera rozkazów.

  Jest to architektura zestawu instrukcji dla mikroprocesora, w którym każda instrukcja może wykonać kilka operacji niskiego poziomu, jak na przykład pobranie z pamięci, operację arytmetyczną, albo zapisanie do pamięci, a to wszystko w jednej instrukcji. Przed powstaniem procesorów RISC, wielu komputerowych architektów próbowało zmostkować lukę semantyczną, aby zaprojektować zestawy instrukcji, które wspierałyby języki programowania wysokiego poziomu przez dostarczenie instrukcji wysokiego poziomu (np. call i return, instrukcje pętli i kompleksowe tryby adresowania). Wszystko to miało na celu połączenie struktur danych i szeregów dostępu do nich w jedną instrukcję procesora. Rezultatem tego były programy o mniejszym rozmiarze i z mniejszą ilością odwołań do pamięci, co w tamtym czasie bardzo ograniczyło koszty pojedynczego komputera.

Z reguły procesory wykonane w architekturze CISC działają wolniej niż procesory RISC (opisane poniżej, o przeciwstawnych założeniach), działające z tą samą częstotliwością.

Przykłady rodzin procesorów o architekturze CISC to:

- System/360;
- VAX;
- PDP-11;
- x86 (procesory Intela i AMD);
- M68000.

- **RISC** (*Reduced Instruction Set Computer*) - nazwa architektury mikroprocesorów, która została przedstawiona pod koniec lat 70 w teoretycznych pracach na uniwersytecie Berkeley oraz w wynikach badań Johna Cocke z Thomas J. Watson Research Center.

  Ówczesne procesory (budowane w architekturze CISC) charakteryzowały się bardzo rozbudowaną listą rozkazów, ale jak wykazały badania, tylko nieliczna ich część była wykorzystywana w statystycznym programie. Okazało się na przykład, że ponad 50% rozkazów w kodzie to zwykłe przypisania (zapis zawartości rejestru do pamięci i odwrotnie).

  Ponadto badania wykazały, że podczas działania programu ok. 26-45% wykonywanych instrukcji to instrukcje wywołania podprocedur lub instrukcje obsługujące pętle, ok. 13-15% to wspomniane przypisania, 7-21% to instrukcje warunkowe (jeśli warunek to …), natomiast reszta stanowi tylko 1-3%.

  W związku z powyższym zaprezentowano architekturę mikroprocesorów, w której uwzględniono wyniki badań. Jej podstawowe cechy to:

  - zredukowana liczba rozkazów do niezbędnego minimum. Ich liczba wynosi kilkadziesiąt, podczas gdy w procesorach CISC sięga setek. Upraszcza to znacznie dekoder rozkazów;
  - redukcja trybów adresowania, dzięki czemu kody rozkazów są prostsze, bardziej zunifikowane, co dodatkowo upraszcza wspomniany wcześniej dekoder rozkazów. Ponadto wprowadzono tryb adresowania, który ogranicza ilość przesłań - większość operacji wykonuje się według schematu: rejestr_C = rejestr_A \ operacja \ rejestr_B ;
  - ograniczenie komunikacji pomiędzy pamięcią, a procesorem. Do przesyłania danych pomiędzy pamięcią a rejestrami służą przede wszystkim dedykowane instrukcje, które zwykle nazywają się *load* (załaduj z pamięci), oraz *store* (zapisz do pamięci); pozostałe instrukcje mogą operować wyłącznie na rejestrach. Schemat działania na liczbach znajdujących się w pamięci jest następujący: załaduj daną z pamięci do rejestru, na zawartości rejestru wykonaj działanie, przepisz wynik z rejestru do pamięci (ten schemat działania przedstawiliśmy w animacji wprowadzającej w tematykę procesorów);
  - zwiększenie liczby rejestrów (przykładowo: 32, 192, 256, podczas gdy np. w architekturze x86 jest zaledwie 8 rejestrów), co również ma wpływ na zmniejszenie liczby odwołań do pamięci;
  - dzięki przetwarzaniu potokowemu (ang. *pipelining*) wszystkie rozkazy mogą być wykonywane w podobnej liczbie kroków, a z punktu widzenia przepustowości procesor może rozpoczynać wykonanie jednej instrukcji w każdym cyklu maszynowym. Pojedyncza instrukcja przechodzi przez kolejne etapy potoku (pobranie, dekodowanie, wykonanie, zapis), ale wiele instrukcji znajduje się jednocześnie w różnych etapach, co pozwala znacząco zwiększyć ogólną wydajność. Zastosowanie superskalarności także na zrównoleglenie wykonywania rozkazów. Dodatkowo czas reakcji na przerwania jest krótszy.

  Za jedne z pierwszych nowoczesnych procesorów zaprojektowanych w oparciu o architekturę RISC uznaje się m.in. projekty IBM 801, Berkeley RISC oraz MIPS.

  Obecnie popularne procesory Intela z punktu widzenia programisty są widziane jako CISC, ale ich rdzeń jest RISC-owy. Rozkazy CISC są rozbijane na mikrorozkazy (ang. *microops*), które są następnie wykonywane przez RISC-owy blok wykonawczy.

Pozostałe metody doboru zestawu instrukcji dla procesora są zdecydowanie mniej popularne, ale wymienimy je dla porządku:

- **VLIW** (*Very Long Instruction Word*) oparty jest na bardzo długiej instrukcji wykonywanej przez procesor, która w zasadzie składa się z wielu prostych instrukcji. Różnica VLIW w stosunku do CISC czy RISC oparta jest głównie na założeniu, że to nie procesor decyduje które z poleceń mogą być wykonane równolegle. Podczas kompilacji instrukcje, które mogą być wykonywane równolegle, są łączone w jedną długą instrukcję i jako taka są przekazywane do wykonania równoległego na procesorze. Dzięki temu upraszcza się architektura procesora;
- **MISC** (*Minimal Instruction Set Computer*) jest to architektura procesora z bardzo małą liczbą podstawowych operacji, które są zgodne z opcode. Takie zestawy instrukcji są częściej oparte na stosie, niż na rejestrach, mając na celu zmniejszenie rozmiaru kodów operacji (opcode). Architektura maszyny stosowej jest właściwie prostsza od kiedy wszystkie instrukcje operują na szczycie większości zapisów w stosie. Rezultatem tego jest mniejszy zestaw instrukcji, mniejsza i szybsza jednostka do dekodowania instrukcji i szybsze wykonanie pojedynczych instrukcji. Wadą takiego podejścia jest to, że trudniej jest wykonywać wiele instrukcji równolegle (częściej daną wejściową do kolejnej instrukcji jest wynik wykonania poprzedniej);
- **ZISC** (*Zero Instruction Set Computer*) jest to technologia oparta na pomysłach wziętych ze sztucznej sieci neuronowej. Układy ZISC składają się z wielu prostych, równoległych „komórek” (neuronów), z których każda porównuje wektor wejściowy z przechowywanym wektorem wzorcowym i zgłasza aktywację przy wystarczająco dobrym dopasowaniu. Sygnał wyjściowy układu jest zwykle wynikiem agregacji (np. ważonej sumy) stanów aktywnych komórek. Równoległość jest kluczem do szybkości układów ZISC – eliminują one konieczność sekwencyjnego przeszukiwania zbioru wzorców w klasycznym sensie. Istotną cechą tych rozwiązań jest również skalowalność: sieć ZISC można rozszerzać przez dodawanie kolejnych układów bez drastycznego spadku wydajności. Praktyczne zastosowania tej technologii koncentrują się głównie na rozpoznawaniu wzorców, systemach bezpieczeństwa, wyszukiwaniu informacji (*data mining*) i podobnych zadaniach, a nie na klasycznym wykonywaniu programów ogólnego przeznaczenia.

Dodatkowe elementy procesora

Do tej pory omawialiśmy modelowy (uproszczony) procesor. W rzeczywistości w opisie współczesnych rozwiązań pojawi się wiele nowych terminów, elementów procesora, których nie ma na żadnym z zamieszczonych rysunków. Postaramy się je przybliżyć, bez szczegółowego wgłębiania się w ich działanie.

Koprocesor

Pierwszym popularnym (a w zasadzie już zawsze obowiązkowym) rozszerzeniem jest koprocesor. Normalny (czyt. przestarzały) procesor komputera wykonywał obliczenia jedynie na liczbach całkowitych - jakiekolwiek operacje na liczbach zmiennoprzecinkowych wymagały wykonywania wielu instrukcji prostych na liczbach całkowitych. Innymi słowy - dodanie 0.1 do 0.1 zajmowało kilkanaście razy więcej czasu niż dodanie 1+1. Od jakiegoś czasu jednak wszystkie procesory mają wbudowaną jednostkę do obliczeń zmiennoprzecinkowych. Jest to odpowiednik ALU, tyle że pozwalający wykonywać podstawowe operacje na zmiennych, które nie są liczbami całkowitymi, co powoduje znaczące przyspieszenie takich obliczeń. We współczesnych procesorach ogólnego przeznaczenia taka jednostka zmiennoprzecinkowa jest zazwyczaj w pełni zintegrowana z rdzeniem procesora, a nie osobnym układem.

Cache

Następnym elementem, który pojawia się wyjątkowo często, jest pamięć podręczna (ang. *cache*) procesora. Normalne pamięci operacyjne (a więc to, co znacie pod pojęciem RAM i co jest opisane dalej w tym podręczniku) są zdecydowanie wolniejsze od samego procesora. Stosowanie w komputerach osobistych pamięci RAM będących w stanie pracować równie szybko jak procesor byłoby bajońsko drogie - zamiast tego stosuje się pamięci podręczne, czyli mechanizm, w którym ostatnio pobierane dane z pamięci operacyjnej są przechowywane w pamięci o zdecydowanie lepszych parametrach szybkościowych. Cache jest elementem właściwie wszystkich podsystemów - nie tylko procesor, ale również np. dostęp do dysku jest buforowany w pamięci RAM. We współczesnych procesorach są zazwyczaj 3 poziomy pamięci cache: L1 (zintegrowana z procesorem), a także L2 i L3 (umieszczone w jednym chipie razem z procesorem). Każdy kolejny poziom cache jest wolniejszą pamięcią, ale za to o większej pojemności.

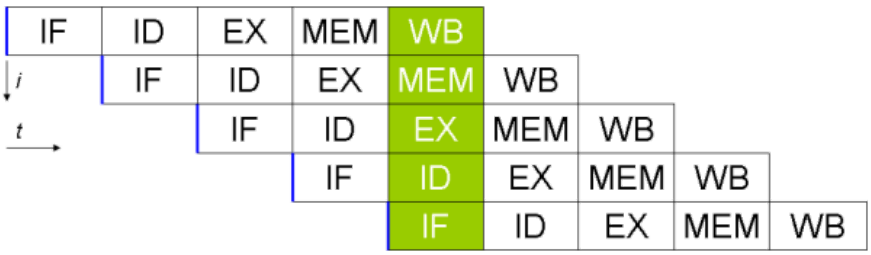
Potokowość

Współczesne procesory często określa się mianem potokowych.

Definicja

Potokowość (ang. *pipelining*) jest techniką budowy procesorów polegającą na podziale logiki procesora odpowiedzialnej za proces wykonywania programu (instrukcji procesora) na specjalizowane grupy w taki sposób, aby każda z grup wykonywała część pracy związanej z wykonaniem rozkazu.

Grupy te są połączone sekwencyjnie i wykonują pracę równocześnie, pobierając dane od poprzedniego elementu w sekwencji. W każdej z tych grup rozkaz jest na innym stadium wykonania. Można to porównać do taśmy produkcyjnej. W uproszczeniu, potok wykonania instrukcji procesora może wyglądać następująco:



1. Pobranie instrukcji z pamięci - ang. *instruction fetch* (IF);
2. Zdekodowanie instrukcji - ang. *instruction decode* (ID);
3. Wykonanie instrukcji - ang. *execute* (EX);
4. Dostęp do pamięci - ang. *memory access* (MEM);
5. Zapisanie wyników działania instrukcji - ang. *store*; *write back* (WB).

W powyższym 5-stopniowym potoku, przejście przez wszystkie stopnie potoku (wykonanie jednej instrukcji) zabiera co najmniej 5 cykli zegarowych. Jednak ze względu na jednoczesną pracę wszystkich stopni potoku, jednocześnie wykonywanych jest 5 rozkazów procesora, każdy w innym stadium wykonania. Oznacza to, że taki procesor w każdym cyklu zegara rozpoczyna wykonanie jednej instrukcji i statystycznie może osiągać przepustowość rzędu jednej instrukcji na cykl zegara. Każdy ze stopni potoku wykonuje mniej pracy w porównaniu do pojedynczej logiki, dzięki czemu może wykonać ją szybciej - z większą częstotliwością - tak więc dodatkowe zwiększenie liczby stopni umożliwia osiągnięcie coraz wyższych częstotliwości pracy.

Podstawowym wrogiem techniki potoku są rozkazy skoku, powodujące w najgorszym wypadku potrzebę przeczyszczenia całego potoku i wycofania rozkazów, które następowały zaraz po instrukcji skoku, a następnie rozpoczęcie zapełniania potoku od początku od adresu, do którego następował skok. Taki rozkaz skoku może powodować ogromne opóźnienia w wykonywaniu programu - tym większe, im większa jest

długość potoku. Dodatkowo szacuje się, że dla modelu programowego x86 taki skok występuje co kilkanaście rozkazów. Z tego powodu niektóre architektury programowe (np. SPARC) zakładały zawsze wykonanie jednego lub większej ilości rozkazów następujących po rozkazie skoku, tzw. skok opóźniony. Stosuje się także skomplikowane metody predykcji skoku lub metody programowania bez użycia skoków.

Wielowątkowość (*Hyperthreading*) i wielordzeniowość (*multicore*)

Inną metodą zwiększania wydajności procesora jest równoległe wykonywanie wielu instrukcji zupełnie niezależnie od siebie. Najprościej wydawało się wykonywać jednocześnie kilka różnych programów (np. Word z pracującym w tle Winampem odgrywającym muzykę), każdy na swoim procesorze.

Jedną z technik wykorzystywanych do tego celu jest przetwarzanie współbieżne (ang. *multiprocessing*, MP). W początkowym wydaniu wykorzystywane było tylko tak zwane symetryczne przetwarzanie współbieżne (SMP), czyli niewielka liczba zupełnie niezależnych procesorów miała dostęp do wspólnej pamięci operacyjnej. Przy takim podejściu, każdy z procesorów musiał być wyposażony w dodatkowe układy sprzętowe zapewniające utrzymywanie spójnej i aktualnej zawartości pamięci (a dokładniej - jej kopii w pamięci cache). Takie podejście sprawdza się w praktyce do systemów o maksymalnie 8 procesorach. W przypadku systemów o większej liczbie procesorów, utrzymywanie spójnej pamięci staje się bardzo trudne i czasochłonne. Dlatego też opracowano architekturę niejednorodnego dostępu do pamięci (ang. *non-uniform memory access*, NUMA) co pozwala na stosowanie tysięcy procesorów w jednym komputerze.

Jednakże NUMA jest architekturą wykorzystywaną głównie przez superkomputery. Natomiast SMP pojawia się w komputerach osobistych lub małych i średnich serwerach. Niektóre procesory serwerowe można montować po 2 lub 4 na specjalnych płytach. Co więcej - normą aktualnie jest obecność na rynku procesorów wielordzeniowych (ang. *multicore*), w których w jednym układzie scalonym znajduje się kilka rdzeni, np. dwa, cztery, osiem i więcej. Historycznym przykładem dwurdzeniowego procesora był Intel Core 2 Duo. Dwa rdzenie - oznacza dwa procesory, każdy z wielopotokowym jądrem i własną pamięcią cache poziomu L1. Czyli, mając jeden chip i jeden wiatrak, mamy komputer wieloprocesorowy 😊

Oprócz możliwości uruchomienia wielu programów naraz, nowoczesne systemy operacyjne umożliwiają także uruchomienie wielu toków wykonania (wątków) w jednym programie. Dokładniej o różnicy między wątkiem a procesem (niezależnym programem) powiemy w lekcji 6. Już teraz jednak zaznaczymy, że różne wątki jednego programu mają tą samą przestrzeń adresową, co znacznie upraszcza budowę procesora, który mógłby wykonywać je równolegle. Taką technologię nazywa się MT (multithreading). Przykładem procesorów z zaimplementowaną tą technologią są procesory Intel Pentium IV HT, a we współczesnych konstrukcjach podobne techniki (wielowątkowość sprzętowa, ang. *simultaneous multithreading*) stosowane są m.in. w wielu procesorach z rodzin Intel Core i oraz Xeon.

W przypadku wielowątkowości warto pamiętać, że procesor HT nigdy nie będzie dwa razy szybszy od odpowiadającego mu procesora jednowątkowego. Wzrost wydajności można zaobserwować jedynie w aplikacjach specjalnie do tego przystosowanych, jak niektóre gry czy Adobe Photoshop, natomiast w większości przypadków komputer z procesorem HT będzie miał porównywalną wydajność do komputera ze zwykłym procesorem.

Równoległe przetwarzanie danych

Zwiększanie wydajności i szybkości obliczeń (co daje w rezultacie szybszy system komputerowy) jest możliwe nie tylko poprzez dodawanie jednostek wykonawczych (ALU) czy też całych procesorów pracujących równolegle. W rzeczywistości do wielu zadań (przetwarzanie grafiki, obliczenia macierzowe, sztuczna inteligencja, statystyka) wykonujemy dokładnie taką samą instrukcję (ciąg instrukcji) na olbrzymiej ilości zmieniających się danych. Jeśli przyjrzymy się budowie procesora oraz przeanalizujemy sposób w jaki działa, szybko możemy dość do wniosku, iż w przypadku takich operacji wystarczy zwielokrotnić stosunkowo mało skomplikowany blok ALU, pozostałe elementy procesora pozostawiając bez zmian, a otrzymamy po jednym pobraniu i zdekodowaniu rozkazu możliwość jego wykonania dla wielu danych, częściowo równolegle, a jeśli nawet szeregowo - to dużo szybciej niż w przypadku pracy w zwykłym trybie.

Taki tryb pracy nazywany jest SIMD (*single instruction*, *multiple data*), a procesory, które go umożliwiają, nazywa się wektorowymi. W obecnych czasach popularność zdobyło jednak trochę inne podejście. Do zwykłego procesora dokładany jest dodatkowy blok funkcjonalny, który jest właśnie takim uproszczonym procesorem wektorowym. Uproszczonym - bo umożliwia wykonanie stosunkowo niewielkiej liczby operacji, zazwyczaj dobieranych tak, by maksymalnie przyspieszyć obsługę multimediów.

Historyczne nazwy handlowe takich układów to MMX (umożliwia wykonywanie operacji jedynie na liczbach całkowitych) czy SSE (operacje wykonywane są na liczbach całkowitych i zmiennoprzecinkowych) u Intela (każdy procesor z tymi oznaczeniami zawiera blok SIMD), 3DNow dla procesorów AMD czy AltiVec dla PowerPC.

W nowszych generacjach procesorów x86 rolę rozszerzeń SIMD pełnią przede wszystkim zestawy instrukcji AVX, AVX2 i AVX-512, natomiast w procesorach ARM podobną funkcję pełnią rozszerzenia NEON oraz SVE.

1.5. Pamięć operacyjna

Co to jest?

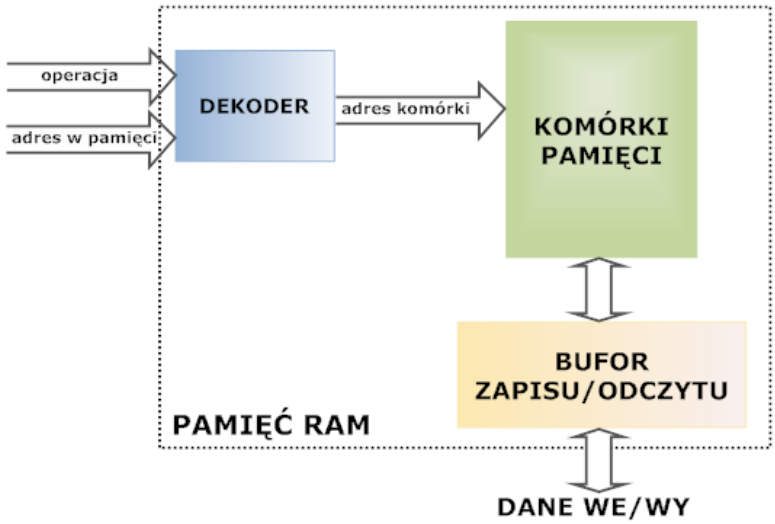
Pamięć operacyjna jest obok procesora jednym z najważniejszych elementów systemu komputerowego i w równym stopniu wpływa na wydajność komputera. Kości pamięci są szybkimi układami logicznymi potrafiącymi, z punktu widzenia użytkownika, przechowywać dane wszelkiego rodzaju. W rzeczywistości przechowywane są jedynie liczby binarne, a ich interpretacja należy do procesora i układów wejścia - wyjścia. Kolejną cechą pamięci operacyjnej jest przechowywanie danych tylko w trakcie działania komputera. Kości pamięci umieszcza się w specjalnych slotach znajdujących się na płycie głównej - tzw. bankach pamięci.

Kości pamięci są szybkimi układami scalonymi zawierającymi miliony (a współcześnie miliardy) komórek przechowujących pojedyncze bity danych.

> **Uwaga**
>
> Z pamięci operacyjnej korzysta procesor umieszczając w niej programy (w tym sam system operacyjny) oraz zapisując i pobierając z niej informacje potrzebne do ich wykonywania.

Pamięć połączona jest z pozostałymi elementami systemu komputerowego za pomocą magistrali systemowej.

Z uwagi na różne sposoby budowy kości pamięci, a dokładniej organizacji pojedynczych komórek z których się składa, adres wykorzystywany przez procesor do określenia konkretnej komórki musi być przetłumaczony na adres fizycznego elementu logicznego będącego częścią układu elektronicznego. Pracę tą wykonuje tzw. dekoder. Oprócz adresu komórki, procesor przekazuje do dekodera również informację o typie operacji, jaką chce na niej wykonać (zapis/odczyt). Jeżeli jako typ operacji został określony odczyt, to po odnalezieniu odpowiedniej komórki pamięci jej wartość jest kopiowana do bufora zapisu/odczytu. W następnym cyklu zegarowym procesor może pobrać pożądaną wartość z bufora. Jeżeli procesor zażądał zapisu informacji do komórki o podanym adresie, to bieżąca wartość bufora zapisu/odczytu jest do niej kopiowana.



Schemat blokowy pokazujący wymianę danych z pamięcią RAM.

Podstawowe cechy pamięci

Pamięć operacyjna (nazywana również pamięcią główną) jest pojemną, szybką i wymazywalną pamięcią ulotną o swobodnym dostępie.

Brzmi strasznie ... Na pierwszy rzut oka powyższa definicja niewiele wyjaśnia. Charakteryzuje ona jednak prawie całkowicie pamięć RAM w naszym komputerze. Postaramy się zatem wyjaśnić pojęcia, które się w niej pojawiły.

Pojemność pamięci to jej główny parametr określający ilość informacji, jaką jest w stanie przechować w jednej chwili czasowej. Pojemność pamięci mierzy się w bajtach (bajt to 8 bitów, podstawowych jednostek pamięci, każdy bit może przyjmować wartość zero lub jeden, czyli bajt może przyjmować 2^8=256 wartości) lub słowach (słowo może być dłuższe niż 8 bitów, to zależy od architektury procesora). Ponieważ dzisiejsze pamięci operacyjne mają pojemności rzędu miliardów bajtów, stosuje się przedrostki Kilo-, Mega- lub Giga-, wszystkie w rozumieniu systemu dwójkowego. A więc 1 KB (Kilo Bajt) to 2^{10} czyli 1024 bajty, podobnie Mega to 1024 Kilo, Giga to 1024 Mega, itd... We współczesnych komputerach osobistych typowa pojemność pamięci operacyjnej wynosi obecnie 8–16 GB, natomiast w komputerach do gier i zastosowań profesjonalnych często spotyka się 32 GB RAM i więcej. W serwerach oraz stacjach roboczych instalowane są zwykle dziesiątki lub setki gigabajtów pamięci, a w dużych systemach – nawet wiele terabajtów. (źródło: Crucial, HP Tech Takes)

Formalnie, zgodnie ze standardami IEC, pojemności oparte na potęgach liczby 2 opisuje się dziś prefiksami *KiB*, *MiB*, *GiB* (kibibajt, mebibajt, gibibajt), a KB/MB/GB odnoszą się do potęg dziesiętnych. W praktyce jednak wciąż często używa się skrótów KB/MB/GB w znaczeniu 1024-krotnym. (źródło: IEC, NIST)

Ogólnie rozumiana **szybkość pamięci** operacyjnej określona jest głównie poprzez częstotliwość z jaką procesor i inne urządzenia komputera mogą z niej korzystać oraz przez szerokość szyny danych. Na szybkość składa się co najmniej kilka elementów. Pierwszym z nich jest czas dostępu do komórki pamięci, czyli czas jaki minie od momentu zażądania informacji przez procesor do momentu pojawienia się tej informacji na wyjściu pamięci. Dla dzisiejszych szybkich pamięci RAM czas dostępu to rząd kilku nanosekund. Drugim elementem wpływającym na szybkość pamięci jest czas cyklu, czyli czas, jaki musi upłynąć pomiędzy dwoma żądaniami dostępu do pamięci. Zwykle jest on nieco dłuższy od czasu dostępu. Częstotliwość pracy dzisiejszych pamięci (np. DDR4, DDR5) jest rzędu setek megaherców zegara bazowego, co dzięki technice podwójnego i wielokrotnego przesyłu danych na takt (DDR, prefetch) przekłada się na efektywne szybkości liczone w kilku-kilkunastu gigatransferach na sekundę (GT/s) i przepustowości rzędu dziesiątek–setek GB/s dla całego systemu.

Pamięć operacyjna jest **wymazywalna**. Oznacza to, że raz zapisana zawartość komórki pamięci może być swobodnie zmieniana. W przeciwnym wypadku pamięć nie miałaby większego pożytku dla procesora, który przecież w trakcie działania przechowuje tam przetwarzane przez siebie dane, w tym wyniki. Dla danych których nigdy się nie zmienia istnieje specjalny typ pamięci niewymazywalnej - ROM (o którym napiszemy więcej za chwilę). ROM nie służy jednak procesorowi do bieżącej pracy lecz zazwyczaj przechowuje podstawowy program uruchamiający elementy składowe systemu komputerowego przed uruchomieniem systemu operacyjnego.

Cechą pamięci operacyjnej jest także jej **ulotność**. Oznacza to, że po zaniku zasilania wszystkie komórki pamięci tracą swoją zawartość, dane w nich przechowywane po prostu znikają (można to porównać do całkowitej amnezji). Przeciwieństwem jest pamięć nieulotna, która zawsze utrzymuje zawartość - nawet bez zasilania.

Ostatnią cechą pamięci wymienioną w pierwszym akapicie jest **swobodny dostęp**. Oznacza to, że każda komórka pamięci ma unikatowy numer, fizycznie wbudowany i przypisany na stałe jej i tylko jej, a czas dostępu do kolejnych komórek nie zależy od poprzednich operacji wykonywanych na pamięci - jest stały. Dzięki temu każda komórka pamięci jest dostępna bezpośrednio dla procesora w tym samym czasie.

Przeciwieństwem dostępu swobodnego jest dostęp sekwencyjny (stosowany chociażby w przypadku taśm magnetycznych, gdzie w celu uzyskania zawartości pewnej komórki pamięci w połowie taśmy należy sekwencyjnie przejść przez wszystkie komórki poprzedzające tą wybraną). Spójrzmy na świat nie-komputerowy. Przykładowo płyta CD daje nam swobodny dostęp do utworów na niej zapisanych, natomiast kaseta magnetofonowa - dostęp sekwencyjny. Szukając t-shirta w szafie na kupce innych t-shirtów mamy do nich dostęp sekwencyjny (najpierw musimy zdjąć te na górze), lecz jeśli umieścilibyśmy każdą koszulkę w oddzielnej szufladzie - mielibyśmy dostęp swobodny, itp.

Specjalnym rodzajem dostępu swobodnego jest dostęp skojarzeniowy (pamięć asocjacyjna). Cechuje się on możliwością porównania i badania zgodności wybranych bitów wewnątrz słowa, przy czym operacja ta wykonywana jest dla wszystkich słów jednocześnie. Słowo jest wyprowadzane na podstawie nie tylko adresu, ale i części swojej zawartości. Taki sposób dostępu stosuje się w pamięciach podręcznych procesora (cache). Podobnie działa też pamięć człowieka.

Pamięć stała

Pamięć **ROM** (ang. *Read Only Memory*), jak sama nazwa wskazuje, jest pamięcią tylko do odczytu. Informacja zapisana w takiej pamięci jest trwała i nie może być zmieniona. Zaletą pamięci ROM w pewnych zastosowaniach jest właśnie jej nieulotność, fakt, że przykładowo program znajduje się w niej cały czas, nigdy nie wymaga ładowania i nie grozi mu żaden wirus.

Swoistą "mutacją" pamięci ROM jest pamięć **EPROM** (wymazywalna programowalna pamięć stała), która ma same zalety - można zmieniać dane w niej zapisane, lecz po zaniku zasilania zawartość EPROM nie niknie (takie połączenie RAM i ROM). Przed operacją zapisu wszystkie komórki zostają skasowane poprzez (zazwyczaj) naświetlenie specjalnego układu (wewnątrz obudowy) promieniowaniem ultrafioletowym. Skoro jest taka doskonała, to dlaczego nie jest używana jako jedyna? Po pierwsze - pamięć EPROM jest droższa od pamięci ROM lub RAM, po drugie - można ją zapisać jedynie kilka razy i po trzecie (i najważniejsze) - jest wolniejsza niż RAM (podobnie jak ROM - odczyt z tej pamięci jest wolniejszy).

Bratem bliźniakiem pamięci EPROM jest **EEPROM** (elektrycznie wymazywalna programowalna pamięć stała). Pamięć ta może być zapisana bez wymazania poprzedniej zawartości, aktualizowane są tylko bajty adresowane. Pamięci EEPROM (i pokrewne technologie) mogą być przeprogramowane ciągle tylko ograniczoną liczbę razy, ponieważ każdy cykl kasowania i ponownego zapisu stopniowo zużywa komórki pamięci - choć tu już mówimy do 10^5 cykli zapisu.

Kolejną powszechnie stosowaną pamięcią stałą jest **pamięć błyskawiczna** (ang. *flash memory*), która wykorzystuje metodę wymazywania elektrycznego. Cała pamięć może być wymazana w ciągu kilku sekund, ponadto możliwe jest wymazanie zawartości tylko niektórych bloków pamięci a nie całego układu. Na bazie takich właśnie pamięci budowane są tzw. pamięci USB czy Pen-Drive, o których więcej powiemy w dalszej części podręcznika. Jednak mimo postępu w dziedzinie konstrukcji wymazywalnych pamięci stałych, ciągle są one zbyt wolne by zastąpić RAM.

Pamięć o swobodnym dostępie (RAM)

Pamięć o dostępie swobodnym **RAM** (ang. *Random Access Memory*) pozwala w stosunkowo łatwy sposób na odczytywanie/zapisywanie danych z/do pamięci. Zapis oraz odczyt odbywają się za pomocą sygnałów elektrycznych. Ważną cechą pamięci RAM jest jej ulotność. Pamięć RAM potrzebuje źródła zasilania, a w przypadku jego braku dane ulegają skasowaniu. Pamięć RAM można podzielić na dynamiczną i statyczną:

- DRAM (ang. *Dynamic Random Access Memory*) - ten rodzaj pamięci o dostępie swobodnym przechowuje każdy bit danych w oddzielnym kondensatorze. Ze względu na to, że nośnikiem informacji są kondensatory sterowane przez trazystory polowe (a dokładnie tranzystory polowe Denarda, ładunek zgromadzony w kondensatorze reprezentuje 0 lub 1), które samoistnie rozładowują się - DRAM wymaga okresowego odświeżania polegającego na odczytaniu i zapisaniu tej samej informacji. Ponadto odczyt z pamięci DRAM jest niszczący (informacja jest kasowana w wyniku rozładowania kondensatora przez współpracujący z nim tranzystor), dlatego też należy powtórnie zapisać odczytane dane tak, aby nie uległy zmianie. Zaletą DRAM jest natomiast niski koszt i możliwość uzyskania wysokiego (jak na tego typu pamięć) stopnia upakowania (dużej ilości bitów pamięci na jednostkę powierzchni). Z tego względu pamięci DRAM są podstawowym typem stosowanym jako pamięć operacyjna.
- SRAM (ang. *Static Random Access Memory*) przechowuje dane tak długo, jak długo włączone jest zasilanie. W odróżnieniu od pamięci typu DRAM nie wymaga okresowego odświeżania. Każdy bit przechowywany jest w pamięci SRAM w układzie zbudowanym z czterech tranzystorów tworzących przerzutnik oraz z dwóch tranzystorów sterujących. Taka struktura umożliwia znacznie szybsze odczytanie bitu niż w pamięci typu DRAM. Pamięci SRAM wykorzystywane są w szybkich pamięciach podręcznych cache wbudowanych w procesor. Tego typu elementy nie wymagają dużych pojemności (gęstość danych w SRAM jest zwykle kilka razy mniejsza niż w DRAM), ale prędkość dostępu jest około 10 razy większa od DRAM.

1.6. Pamięć masowa

Główną wadą omawianych w poprzednim segmencie pamięci operacyjnych jest ich ulotność oraz stosunkowo niewielka pojemność. Główną zaletą - duża szybkość działania. Jednakże istnieje duża grupa zastosowań, gdzie szybkość działania pamięci jest mniej istotna, natomiast ważne jest trwałe przechowywanie zapisanych danych, czy też znacznie większa pojemność. Często nie bez znaczenia jest również fakt "przywiązania" pamięci operacyjnej tylko do jednego systemu komputerowego, co oznacza brak możliwości przekazania danych w niej zapisanych innej osobie.

To trochę tak, jak z naszą pamięcią. Mózg w sposób bezpośredni przetwarza jedynie to, co ma w swoich szarych komórkach - to one stanowią naszą "pamięć operacyjną". Jednak zdajemy sobie świetnie sprawę, że nie zdołamy zapamiętać wszystkiego oraz że po jakimś czasie większość faktów zapominamy. Dlatego potrzebujemy jakiegoś trwałego nośnika informacji, nośnika wiedzy. Taką rolę dla człowieka pełnią notatki, książki, dokumenty, fotografie - czyli ogólnie biblioteka, miejsce gdzie zawsze możemy umieścić nieprzydatną dla nas w tej chwili wiedzę i później - jak będzie taka potrzeba - w każdej chwili do niej wrócić.

Rolę takiej biblioteki dla komputerów pełnią **pamięci masowe** - nieulotne i przeznaczone do długotrwałego przechowywania dużej ilości danych. W zależności od zjawiska fizycznego wykorzystywanego do zapisu informacji, pamięci masowe dzieli się zazwyczaj na trzy główne kategorie.

Pamięci magnetyczne

W pamięci tego typu informacja jest przekazywana przy wykorzystaniu zjawisk magnetycznych. Pierwszym urządzeniem wykorzystującym efekt magnetyczny do zapisu był tzw. "drutofon". W 1898 r. duński inżynier Valdemar Pulsen opracował zapamiętywanie ludzkiej mowy na pierwowzorze magnetofonu – dźwięki zapisywano na drucie (strunie fortepianowej) magnesowanym przez elektromagnes, w którym prąd był wzbudzany przez mikrofon telefoniczny przetwarzający dźwięki na sygnał elektryczny. Potem, w trakcie 2 wojny światowej w Niemczech wynaleziono taśmę magnetyczną - cienki celuloidowy pasek pokryty warstwą tlenku żelaza, nawinięty na szpulę.

Rozwój pamięci magnetycznych nastąpił z chwilą pojawienia się komputerów. Szybko zauważono, że idealnie nadaje się ona do zapisu danych binarnych. Logicznemu zeru i jedynce mogą odpowiadać dwa stany różniące się zwrotami namagnesowania.

Pierwsze podejście było związane z pamięciami rdzeniowymi. Tego typu pamięć zawierała tysiące rdzeni magnetycznych o średnicy 2 mm wykonanych z ferrytu. Była ona bardzo szybka (jak na owe czasy) - choć nie pozwalała na zbyt wysoką gęstość zapisu (niewielka pojemność). Magnetyczne pamięci rdzeniowe zostały wyparte przez twarde dyski – o nich za chwilę powiemy więcej.

Równolegle wykorzystywano taśmy magnetyczne ze względu na ich ogromną pojemność. Jednak z powodu ich niskiej szybkości działania oraz sekwencyjnego dostępu do danych, dziś wykorzystuje się je głównie w systemach do tworzenia kopii zapasowych i długoterminowej archiwizacji danych (tzw. streamerach), szczególnie w dużych centrach danych i rozwiązaniach chmurowych, gdzie liczy się bardzo niska cena przechowywania informacji przez wiele lat (źródło: np. IBM: Magnetic tape data storage). Natomiast we wszystkich pozostałych zastosowaniach zostały wyparte albo przez dyski twarde, albo pamięci optyczne.

Dysk magnetyczny ma kształt okrągłej płyty - sztywnego krążka o podłożu z metalu lub szkła, na którym osadzona jest warstwa materiału magnetycznego (kiedyś tlenek żelazowy, z czasem stopy metali, w których główny składnik to kobalt) będąca nośnikiem informacji. Dane są na nim zapisywane i odczytywane za pomocą głowicy, którą stanowi w tym wypadku cewka elektryczna. W czasie operacji odczytu/zapisu wiruje dysk, zaś głowica pozostaje nieruchoma.

Podczas zapisu wykorzystywane jest pole magnetyczne wytwarzane przez prąd płynący w cewce. Po nadejściu żądania zapisu, do głowicy wysyłane są impulsy, które powodują czasowe zmiany kierunku pola magnetycznego przez nią generowanego. Zmiany te powodują analogiczne zmiany kierunku namagnesowania powierzchni dysku, która znajduje się w danej chwili pod głowicą (przypominam, że dysk cały czas wiruje). Odczyt polega na wykorzystaniu zjawiska indukowania przepływu prądu elektrycznego pod wpływem pola magnetycznego powierzchni dysku - tym razem głowica "słucha" zamiast "mówić".

Pierwszym szeroko wykorzystywanym typem dysków magnetycznych były elastyczne dyskietki (aktualnie wyparte przez pamięci optyczne lub półprzewodnikowe) o kilku rozmiarach - 8 cali, 5.25 cala oraz 3.5 cala. Dyskietkę zawsze można było wyjąć z napędu, zamienić na inną i zabrać ze sobą - głowica była natomiast na stałe osadzona w napędzie. Jednakże ze względu na ryzyko uszkodzeń mechanicznych oraz obecność zanieczyszczeń, dyskietki nie mogły osiągnąć zbyt wysokiej pojemności.

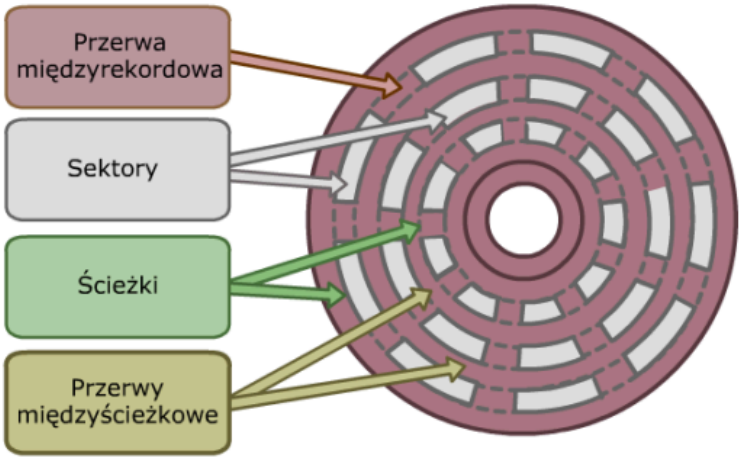
Inne podejście zastosowano w przypadku dysków twardych. Tam jeden bądź więcej talerzy zamknięto wraz z głowicami, silnikami i elektroniką sterującą w jednej, hermetycznej i nieotwieralnej obudowie, co pozwoliło na znaczne zwiększenie gęstości zapisu oraz szybkości działania dysku. Współcześnie prędkość liniowa na dysku przekracza 100 km/h, a odległość unoszącej się nad nim głowicy – 50 milionowych części milimetra. Głowica unosi się nad powierzchnią magnetyczną na poduszce powietrznej generowanej przez obracający się z prędkościami rzędu kilku tysięcy obrotów na minutę dysk. To powoduje, że dyski twarde są dość wrażliwe na wstrząsy w trakcie pracy oraz w zasadzie nierozbieralne (rozhermetyzowanie dysku uniemożliwia jego dalsze wykorzystanie). Aby dalej zwiększać pojemność bez zmiany rozmiaru fizycznego dysku, producenci stosują zaawansowane techniki zapisu, takie jak *shingled magnetic recording* (SMR) czy *heat-assisted magnetic recording* (HAMR), pozwalające upakować więcej bitów na tej samej powierzchni talerza (źródło: np. Seagate: What is SMR?, Seagate: What is HAMR?). Współcześnie dyski twarde wciąż oferują bardzo duże pojemności przy niskim koszcie za gigabajt, dlatego pozostają

podstawowym nośnikiem danych w serwerach i wielu komputerach stacjonarnych. Jednak pod względem szybkości dostępu i transferu danych zostały wyraźnie wyprzedzone przez dyski półprzewodnikowe SSD, które mogą być nawet kilkadziesiąt razy szybsze, choć nadal są droższe w przeliczeniu na jednostkę pojemności (źródło: np. Seagate: Hard Drives vs. SSDs, HP: SSD vs HDD).

Organizacja i formatowanie danych

Dane na dysku twardym zapisywane są w postaci tzw. **ścieżek**. Cała powierzchnia dysku jest podzielona na równo rozłożone pierścienie. Każda ścieżka ma taką samą szerokość jak głowica. Pomiędzy każdymi dwoma ścieżkami jest zawsze zostawiony pewien niewielki odstęp, dzięki czemu błędy wynikające z niewłaściwego ustawienia głowicy lub interferencji pola magnetycznego są minimalizowane.

Historycznie, aby uprościć układy elektroniczne współpracujące z głowicą, przyjmowano, że na każdej ścieżce przechowywana jest taka sama liczba bitów, a wielkość tą nazywa się **gęstością** i wyraża w bitach na cal. W nowoczesnych dyskach twardych stosuje się jednak technikę *zone bit recording* (ZBR), w której zewnętrzne ścieżki zawierają więcej sektorów niż wewnętrzne, co pozwala zwiększyć całkowitą pojemność dysku (źródło: np. Zone bit recording).



Organizacja dysku magnetycznego

Później (w wykładzie poświęconym systemom plikowym) dowiecie się, iż dane na dysku zapisywane są praktycznie zawsze blokami w celu zwiększenia wydajności transferu oraz umożliwienia łatwego wyszukiwania, dodawania i kasowania danych. Takie bloki danych na dysku nazywa się **sektorami**. Na ścieżkę przypadać może wiele sektorów o zmiennej lub stałej długości. Poszczególne sektory wewnątrz ścieżek na dysku są identyfikowane dzięki dodatkowym danym kontrolnym, umieszczanym pomiędzy nimi - w ten sposób oznacza się początek i koniec każdego sektora względem sektora startowego.

Współczesny dysk twardy jest opisywany kilkoma podstawowymi parametrami. Pierwszym z nich jest pojemność dysku, zazwyczaj podawana w GB oraz jako maksymalna możliwa do uzyskania pojemność, bez uwzględniania struktur kontrolnych. Dlatego też nie należy się dziwić, jeśli dysk o pojemności 200 GB w systemie jest pokazywany jako 180 GB.

Kolejnym, jednym z ważniejszych parametrów, jest szybkość dysku, którą określają dwa czynniki:

1. **Tempo przesyłania** (ang. *transfer rate*) - oznacza szybkość, z jaką dane są transmitowane pomiędzy dyskiem a komputerem.
2. **Czas ustalania położenia głowicy** (ang. *positioning time*, nazywany też ang. *random access time*). Na wielkość tą składają się:
    - *czas wyszukiwania* (ang. *seek time*), czyli czas przesuwania głowicy do odpowiedniego cylindra (cylinder jest to kilka ścieżek umieszczonych fizycznie nad sobą na kilku talerzach dyskowych);
    - *opóźnienie obrotowe*, czyli czas potrzebny na to, aby odpowiedni sektor, przeszedł nad głowicą.

Dyski magnetyczne współpracują zarówno z głowicami nieruchomymi (jedna głowica zapisu/odczytu na jedną ścieżkę) oraz z głowicami ruchomymi - istnieje wtedy tylko jedna głowica zapisu/odczytu na talerz (np. współczesne dyski twarde).

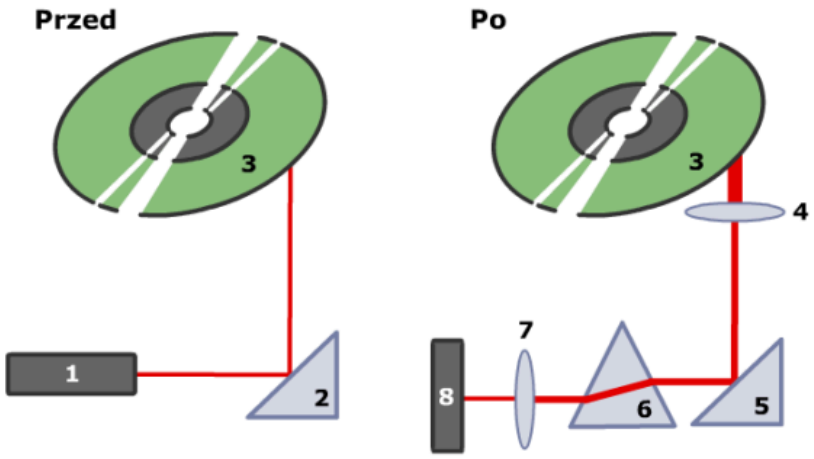
Pamięci optyczne

Prace nad napędem CD rozpoczęte zostały w roku 1978. Pierwszy napęd powstał dzięki firmie Philips, ale odtwarzacz CD-ROM w dzisiejszej postaci zaprojektowany został przez firmy Philips i Sony. W połowie lat 80 minionego wieku napędy weszły do szerszej dystrybucji, zaś lata 90 przyniosły im niezwykłą popularność głównie dzięki stosunkowo niskiej cenie oraz dużej pojemności jaką oferowały płyty CD-ROM. Przy czym należy tu zaznaczyć, że CD-ROM jest technologią adoptowaną dla potrzeb systemów komputerowych - jej pierwotnym przeznaczeniem było przechowywanie i udostępnianie muzyki.

Płyty CD i CD-ROM są wykonane w ten sam sposób - dysk składa się z czterech warstw: warstwy nośnej z poliwęglanu (plastik), warstwy barwnika (topi się podczas zapisu), warstwy dobrze odbijającej światło (np. aluminium) i lakierowej warstwy ochronnej.

Płyta CD posiada spiralny rowek prowadzący (ang. *groove*), który naprowadza laser, wskazuje mu drogę. Informacja zarejestrowana cyfrowo nanoszona jest w postaci małych zagłębień (ang. *pit*) na powierzchnię odbijającą. Pierwszy egzemplarz wykonuje się to za pomocą dobrze zogniskowanej wiązki laserowej o dużej mocy (podczas zapisu barwnik wytapia się, odsłaniając odbijającą światło powierzchnię). Tak powstaje dysk wzorcowy, który służy jako matryca do tłoczenia kopii. Powierzchnia z naniesioną informacją jest zabezpieczona bezbarwnym lakierem.

Odczyt z płyty odbywa się z wykorzystaniem lasera małej mocy wbudowanego w napęd CD. Laser świeci na wirującą płytę (wprawianą w ruch przez silniczek napędu), pokrytą przezroczystą warstwą lakieru. Zmiana natężenia odbitego światła, zarejestrowana przez komórki światłoczułe, oznacza, że promień napotkał zagłębienie, a więc daną, która następnie zostaje przekonwertowana na sygnał cyfrowy. Dokładniej możecie prześledzić to na rysunku poniżej:



Zasada działania pamięci optycznych - schemat

1. Dioda laserowa (1) emituje na zwierciadło (2) wiązkę promieni podczerwonych o niskiej energii.
2. Mikrosilnik krokowy (serwonapęd), przesuwając zwierciadło (2) na rozkaz mikroprocesora, ustawia wiązkę świetlną we właściwej odległości od środka płyty (na poszukiwanym wycinku ścieżki) (3).
3. Wiązka promieni świetlnych po odbiciu od płyty (3) jest zbierana i skupiana przez soczewkę (4) znajdującą się pod płytą, następnie odbijana od zwierciadła (5) i kierowana do układu pryzmatów (6).
4. Układ pryzmatów (6) kieruje wiązkę promieni świetlnych na następną soczewkę skupiającą (7).
5. Soczewka (7) kieruje wiązkę promieni na fotodetektor (8), który zamienia ją na ciąg impulsów elektrycznych. Następnie impulsy te są dekodowane i przesyłane do komputera.

W zależności od tego, czy dany **pit** na płycie odbija światło czy nie - wiązka promieni będzie zmieniała swoje natężenie na fotodetektorze pomiędzy dwoma stanami: niskim i wysokim. Niski odpowiada wartości logicznej 0, wysoki 1. I oto mamy kolejną metodę na przechowywanie danych binarnych.

Aby umożliwić stałą prędkość odczytu danych przez laser, stosuje się napędy, które obracają krążki:

- **ze stałą prędkością kątową** (ang. *constant angular velocity - CAV*).
  Dysk jest podzielony na pewną liczbę sektorów oraz na szereg koncentrycznych ścieżek. Zaletą tego rozwiązania jest łatwe adresowanie z użyciem ścieżki i sektora. Także przesunięcie głowicy pod określony adres wymaga niewielkiego jej ruchu i oczekiwania, aż szukany sektor znajdzie się nad głowicą. Na ścieżkach wewnętrznych i zewnętrznych można umieścić taką samą liczbę danych, dlatego też metoda ta nie znalazła szerszego zastosowania w napędach CD-ROM;
- **ze stałą prędkością liniową** (ang. *constant linear velocity - CLV*).
  Dysk obraca się więc wolniej podczas odczytu danych z zewnętrznych sektorów płyty, zaś w pobliżu jej środka "przyspiesza". Taka konstrukcja sprawia, że pojemność ścieżek znajdujących się na obrzeżach dysku znacznie wzrasta.

Dane na dysku CD-ROM, podobnie jak dla dysków magnetycznych, są zorganizowane w bloki. Bloki te składają się z pól:

- *sync* (synchronizacja) - 12 bajtowe pole identyfikujące początek bloku (składa się z bajtu 0, 10 bajtów 1 i bajtu 0);
- *header* (nagłówek) - 4 bajtowe pole zawierające adres bloku i bajt trybu. Tryb 0 to czyste pole danych, tryb 1 - wykorzystanie kodu korekcyjnego dla 2048 bajtów danych, tryb 2 to 2236 bajtów danych bez kodu korekcji;
- *data* (dane) - tu zaczyna się użyteczna dla użytkownika przestrzeń. Standardowo pojemność jednego bloku jest ograniczona do 2048 bajtów;
- *auxiliary* (pomocnicze) - dodatkowe dane użytkownika w trybie 2 lub 288 bajtowy kod korekcyjny w trybie 1.

Pierwotnie napędy czysto optyczne (takie jak opisany powyżej CD) nie umożliwiały użytkownikowi zapisu danych, mogły zawierać jedynie dane wytłoczone na nich raz na zawsze. Później pojawiły się napędy, które umożliwiały jednokrotny zapis czystej płyty. Zasada działania w tym przypadku jest analogiczna jak przy odczycie, zmienia się jedynie moc lasera. Podczas odczytu laser pracuje z niewielką mocą, która nie pozwala na uszkodzenie powierzchni nośnika. W trakcie zapisu laser generuje impulsy o dużej mocy - które wypalają pity na płycie. Technologia RW (płyty CD-RW, DVD-RW, BD-RE) umożliwia skasowanie raz zapisanej informacji i powtórne wykorzystanie nośnika. Warto jednak zauważyć, że w typowych komputerach osobistych napędy optyczne są coraz rzadsze – wiele nowych laptopów nie ma ich w ogóle, a

dystrybucja oprogramowania i multimediów przeniosła się do Internetu. Nośniki optyczne zachowują jednak znaczenie w zastosowaniach archiwizacyjnych oraz przy dystrybucji filmów (źródło: np. Fujifilm: Optical Data Storage, raporty rynkowe dotyczące napędów optycznych). W uproszczeniu efekt RW osiągnięto poprzez wykorzystanie jako warstwy odblaskowej materiałów, które po krótkim ogrzaniu nie odbijają światła, przy nieco dłuższym (do wyższej temperatury) zaczynają odbijać.

Pamięci magnetooptyczne

Połączeniem omówionych wcześniej technik są pamięci magnetooptyczne. W ich przypadku odczyt danych odbywa się przy pomocy światła (lasera), natomiast do zapisu wykorzystywane są efekty magnetyczne. Najczęściej jako warstwę odblaskową w tego typu pamięciach stosuje się materiały, dla których współczynnik odbicia światła zmienia się wraz z kierunkiem namagnesowania.

Zastosowanie takiej pamięci wymaga budowy złożonej głowicy optyczno-magnetycznej, co spowodowało że napędy magnetooptyczne nie zdobyły szerszej popularności i nie będziemy im poświęcali w tym podręczniku więcej uwagi.

Pamięci półprzewodnikowe

W ostatnich latach coraz popularniejsze staje się wykorzystanie jako pamięci masowej pamięci półprzewodnikowych. Najczęściej są to pamięci typu Flash omówione w poprzednim segmencie, umożliwiające wielokrotny zapis i odczyt poszczególnych komórek. Ich potężną zaletą, w porównaniu z wcześniej omówionymi pamięciami optycznymi i magnetycznymi, jest brak jakichkolwiek mechanicznych ruchomych elementów. Wcześniej mieliśmy do czynienia zawsze z obracającym się z dużą szybkością nośnikiem - co zawsze jest źródłem hałasu, wydzielania się ciepła oraz zużycia znacznych ilości energii przez silnik napędzający. Ponadto nośniki narażone są na mechaniczne uszkodzenia i ogólnie - mniej odporne na zużycie i uszkodzenia.

Pamięci półprzewodnikowe pozbawione są wspomnianych wad. Są znacznie mniejsze, pobierają o rząd wielkości mniej energii i pracują zupełnie bezgłośnie. Problemem w ich przypadku jest nadal wyższy koszt w przeliczeniu na MB pojemności niż w przypadku klasycznych dysków magnetycznych, choć różnica ta z roku na rok maleje (źródło: np. Lexar: SSD vs HDD). Pod względem szybkości działania jest odwrotnie – nowoczesne dyski SSD oparte na pamięci flash są wielokrotnie szybsze niż dyski twarde, ustępują natomiast pamięci operacyjnej RAM i dlatego nie zastępują jej w komputerze (źródło: np. HP: SSD vs HDD).

Czasem też musicie uważać na nazewnictwo. Współcześnie w komputerach osobistych coraz częściej spotyka się tak zwane dyski SSD (ang. Solid State Drive), które tak naprawdę są pamięcią flash - opakowaną w pudełko dysku i wyposażoną w jego interfejsy. Żadnego fizycznego dysku tam nie ma ... ale ciągle bardzo częstp nazywa się takie rozwiązanie po polsku dysk półprzewodnikowy - no cóż, siła przyzwyczajeń wygrywa w tym przypadku z logiką.

W języku angielskim nazwy też nie są dobrane idealnie, lecz tam wspólnym mianownikiem nie jest dysk, a napęd. I dlatego mamy, w historycznej kolejności - Floppy Disc Drive, Hard Disc Drive oraz Solid State Drive, przy czym solid state pochodzi od solid state physics i jest potocznie kojarzony ze stosowaniem tranzystorów i układów scalonych.

Tryb dostępu do danych

Jedną z wymaganych cech współczesnych pamięci masowych jest bezpośredni dostęp do danych na nich umieszczonych. Oznacza to, że zapis i odczyt realizowany jest przez ten sam mechanizm (głowicę, laser) oraz dane są zapisywane w blokach. Poszczególne bloki mają unikatowy adres oparty na lokalizacji fizycznej, a dostęp odbywa się w dwóch etapach. Najpierw następuje pozycjonowanie głowicy w najbliższym otoczeniu danych (na odpowiedniej ścieżce lub jej fragmencie). W drugim etapie następuje sekwencyjne poszukiwanie, liczenie lub oczekiwanie w celu osiągnięcia szukanej lokalizacji (obrót talerza lub płyty do odpowiedniej pozycji). Dlatego całkowity czas dostępu do informacji jest zmienny i zależy od aktualnego stanu urządzenia.

Z tego powodu nie można traktować pamięci masowych (z wyłączeniem półprzewodnikowych) jako pamięci o dostępie swobodnym (jak np. RAM) - postulat o jednakowym czasie dostępu do każdej komórki w tym wypadku nie jest spełniony. W zamian za to nazywamy je pamięcią o dostępie bezpośrednim, w odróżnieniu od np. taśmy magnetycznej, gdzie możliwy był jedynie dostęp sekwencyjny. Pamięci półprzewodnikowe (SSD, pamięci flash) nie mają części ruchomych, dlatego czas dostępu do dowolnego bloku danych jest w praktyce prawie stały – z punktu widzenia systemu operacyjnego można je traktować jak pamięć o dostępie swobodnym, choć nadal są one znacznie wolniejsze niż pamięć operacyjna RAM.

1.7. Przerwania

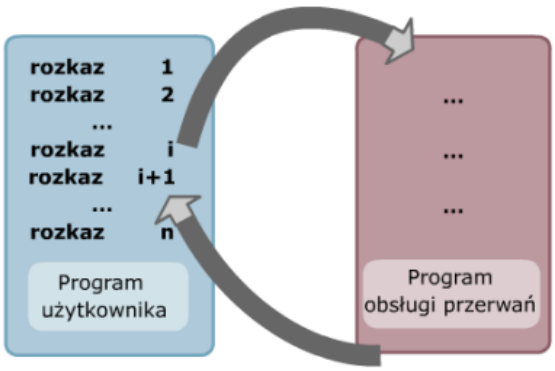
Dla wszystkich, którzy zetknęli się ze współczesnymi systemami komputerowymi, zupełnie normalną cechą jest możliwość wykonywania wielu zadań w jednym czasie. Tak naturalną, że zupełnie nie zastanawiamy się jak to jest możliwe ... a pomyślmy przez chwilę...

Rozważmy bardzo prosty przykład - zlecamy naszemu komputerowi obliczenia, które trwają dajmy na to 10 minut. W międzyczasie ruszamy myszką i ... i wskaźnik na ekranie poruszył się! Biorąc pod uwagę fakt, że o zawartości ekranu decyduje procesor i że nawet w przypadku pojedynczego rdzenia procesor może fizycznie wykonywać tylko jeden rozkaz w danej chwili, wydaje się to na pierwszy rzut oka niemożliwe...

Oczywiście, nie był (i nie jest) to żaden cud. Istnieje kilka mechanizmów pozwalających na przerwanie procesorowi wykonywania aktualnego zadania i zlecenie mu natychmiastowego wykonania małej roboty na boku. Głównym i najważniejszym z tych mechanizmów są *przerwania*. Sama nazwa też nie jest przypadkowa - sygnalizuje, że przerwania mają w zwyczaju przerywać normalną pracę programów.

Dokładnie tak, jak dzieje się w przypadku człowieka, gdy podczas pracy zadzwoni telefon. Dajmy na to, że piszemy ręcznie list do przyjaciela/przyjaciółki. W pewnym momencie słychać dzwonek telefonu - odkładamy więc pióro, podnosimy słuchawkę i zajmujemy się przez chwilę rozmową telefoniczną. Gdy rozmowa się skończy, bierzemy do ręki pióro i wracamy do wcześniej wykonywanej pracy kontynuując zdanie które przerwaliśmy.

Jak w ogólnym zarysie działają przerwania? Wyobraźmy sobie, że pewne urządzenie zewnętrzne jest gotowe do obsługi (czyli np. przyjmowania danych od procesora, czy też - z wprowadzającego przykładu - zmieniło się położenie myszki i chce ona poinformować o tym procesor). Wtedy moduł wejścia/wyjścia (ang. *I/O, Input/Output*) tego urządzenia wysyła sygnał **żądania przerwania** do procesora (*dzwoni dzwonek telefonu*). W odpowiedzi procesor zawiesza działanie bieżącego programu (*odkładamy pióro*) i wykonuje skok do **programu obsługi przerwania** (ang. *interrupt handler*) obsługującego to urządzenie (*rozmawiamy przez telefon*). Po obsłużeniu urządzenia następuje powrót do programu, który został przerwany (*wracamy do pisania*). Żeby opisany wyżej schemat działania był możliwy, potrzebna jest jeszcze współpraca z systemem operacyjnym: współcześnie to jądro systemu operacyjnego (kernel) odpowiada za obsługę przerwań i decyduje, które procesy i sterowniki urządzeń zostaną o nich poinformowane. Więcej o tym przeczytacie w trakcie omawiania systemów operacyjnych.



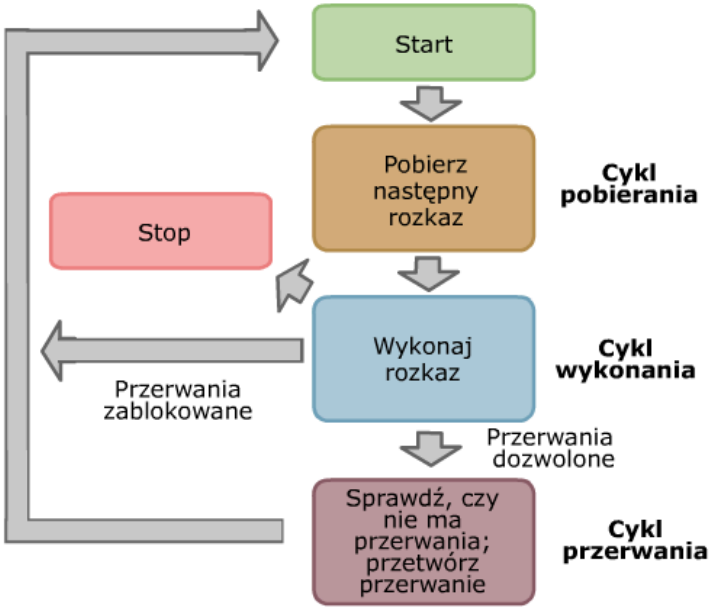
Przekazywanie sterowania przez przerwania

Ponieważ przerwania powinny być obsługiwane szybko, system posługuje się tablicą wskaźników do procedur obsługujących przerwania. Owa tablica nazywana jest **wektorem przerwań** (ang. *interrupt vector*) i jest indeksowana jednoznacznym numerem urządzenia.

Występowanie przerwań modyfikuje nieco schemat cyklu rozkazowego. Do cyklu rozkazu dodawany jest **cykl przerwania**. W trakcie cyklu przerwania procesor sprawdza, czy nie pojawiły się jakieś przerwania. Jeśli przerwania nie są wykonywane, procesor pobiera następny rozkaz danego programu. Jeśli natomiast wystąpi zgłoszenie przerwania procesor:

1. Zawiesza wykonywanie bieżącego programu i zachowuje jego kontekst, czyli przesyła do specjalnie wydzielonego obszaru pamięci zawartość rejestrów oraz adres następnego rozkazu (zawartość licznika rozkazów). Ten wydzielony obszar pamięci nazywa się **stosem systemowym** i jest zarządzany przez jądro systemu operacyjnego.
2. Licznik rozkazów jest ustawiany na początkowy adres programu obsługi przerwań.

Następnie procesor pobiera pierwszy rozkaz z programu obsługi przerwań i zaczyna go wykonywać.



Cykl rozkazu z przerwaniami

Warto wspomnieć, że zaawansowane architektury przerwań pozwalają na obsługę nowego przerwania przed zakończeniem poprzedniego. System operacyjny korzysta ze specjalnego schematu priorytetów, który określa jak ważna jest obsługa danego urządzenia. Przerwanie o wyższym priorytecie będzie obsługiwane również wtedy, gdy jest w trakcie wykonywane przerwanie o priorytecie niższym, zaś przerwania o tym samym lub niższym priorytecie będą *maskowane* (ang. *masked interrupt*). Szczegóły działania priorytetów i maskowania przerwań zależą od konkretnej architektury procesora (np. x86, ARM, różne mikrokontrolery), ale ogólna idea pozostaje taka sama.

1.8. Operacje wejścia / wyjścia

Mówiliśmy już o mózgu komputera (procesorze), następnie o pamięci, przyszła zatem pora na odpowiednik zmysłów - czyli układy wejścia/wyjścia, przy pomocy których procesor komunikuje się ze światem zewnętrznym. Nie będziemy teraz omawiali szczegółowo zasady działania takich układów, ograniczymy się do omówienia sposobu przesyłania danych z urządzeń wejścia / wyjścia do komputera.

Na początek zajmiemy się fizycznymi aspektami transmisji sygnałów, a następnie omówimy tryby pracy procesora i pamięci w trakcie obsługi sygnałów zewnętrznych. Zanim jednak zaczniemy - ważna uwaga:

> **Uwaga**
>
> wszystkie sygnały przetwarzane przez procesor i wszystkie wartości, które są przechowywane w pamięci, muszą mieć postać cyfrową.

Nasz świat nie jest cyfrowy - jest analogowy. Temperatura powietrza jest wartością, która zmienia się w sposób ciągły, co oznacza że pomiędzy dwie dowolne, ale różne wartości zawsze da się wstawić jeszcze jedną, np.: pomiędzy 20 a 21 stopni jest 20,5, pomiędzy 20 a 20,5 jest np. 20.1, pomiędzy 20 a 20,1 jest 20,09 i tak dalej - w nieskończoność.

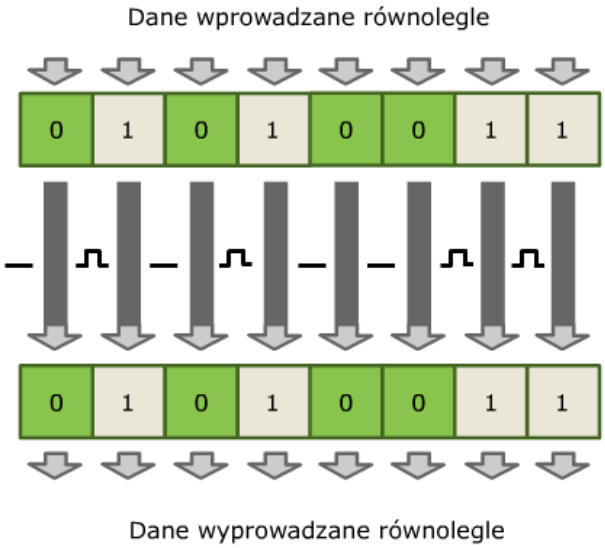
Komputer nie rozumie wartości analogowych. Dla niego istnieją tylko wartości zdyskretyzowane. Na czym to polega? Umówmy się przykładowo, że 0 stopni Celsjusza przypiszemy wartość 0, 100 stopniom Celsjusza przypiszemy wartość 1000. Teraz 20 stopni to liczba 200, 20.5 stopnia to 205, ale już zarówno 20.1 jak i 20.09 to ta sama liczba - 201. Komputer stracił zdolność rozróżniania tych liczb. W większości przypadków to nic nie szkodzi, przecież my też nie odróżniamy 20 stopni od 20.1 - i jakoś żyjemy ;) Tyle tylko, że nasze zmysły działają w sposób ciągły i w ten sposób analizują przychodzące informacje, natomiast w przypadku komputerów najpierw wartość analogowa (ciągła) musi zostać zamieniona na cyfrową (dyskretną). Takiego przetworzenia dokonują przetworniki A/C (ang. *ADC – Analog-to-Digital Converter*). Wynik pracy komputera, jeśli ma zostać zaprezentowany w postaci analogowej (np. muzyka którą generuje karta dźwiękowa), także musi zostać poddany przekształceniu - tym razem odwrotnemu, z sygnału cyfrowego na analogowy. Do tego służą przetworniki C/A (ang. *DAC – Digital-to-Analog Converter*).

Przetworniki są urządzeniami elektronicznymi i nie będziemy się tutaj zagłębiać w ich budowę. Natomiast z powyższych rozważań jeden fakt jest dla nas istotny - wewnątrz komputera wszystkie przesyłane wartości mają postać cyfrową (zestaw bajtów, z których każdy składa się z 8 zer lub jedynek). I jak taką liczbę przesyła się pomiędzy procesorem, pamięcią i urządzeniami zewnętrznymi?

Rodzaje transmisji

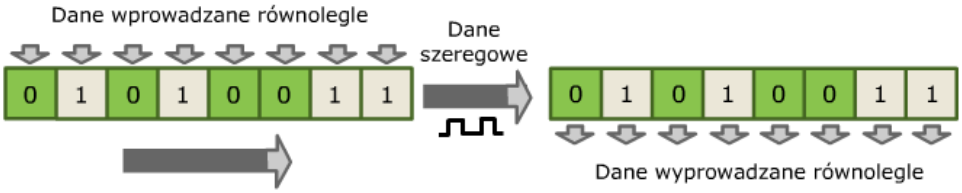
Najbardziej podstawowym z podziałów jest podział transmisji sygnału na równoległą i szeregową.

Transmisja równoległa polega na przesyłaniu wszystkich pól słowa danych jednocześnie, czyli przykładowo; całego bajtu, dwóch bajtów, 64 bitów, itp.. w zależności od standardu interfejsu komunikacyjnego. Ze względu na fakt, iż sprzęt komputerowy w naturalny sposób wspiera taki sposób wprowadzania/wyprowadzania danych (procesor operuje na całym rejestrze a nie na poszczególnych bitach w nim umieszczonych), transmisja równoległa jest bardzo popularna. Ponadto, dzięki zrównolegleniu przesyłu danych, uzyskujemy znaczny przyrost prędkości przesyłu. Udogodnienia te wiążą się jednak z koniecznością stosowania dużej liczby fizycznych kanałów transmisyjnych (np. przewodów). Klasycznym przykładem takiego rozwiązania był równoległy interfejs Centronics używany kiedyś do podłączania drukarek (wymagał 18 przewodów), a także taśmy do podłączania dysków twardych zgodne ze standardem równoległym EIDE, mające łącznie 80 żył. Współcześnie interfejsy tego typu zostały w komputerach osobistych w dużej mierze wyparte przez szybsze i prostsze w okablowaniu interfejsy szeregowe (np. SATA, PCI Express).



Transmisja równoległa

Transmisja szeregowa polega na sekwencyjnym przesyłaniu danych bit po bicie. W tym celu, przed przesłaniem informacja jest wprowadzana do tzw. **rejestru przesuwnego** styku szeregowego, który następnie bit po bicie wysyła ją na wyjście układu.



Transmisja szeregowa

Zaletą tego rodzaju transmisji jest możliwość przesyłu danych na dużo większą odległość niż w przypadku transmisji równoległej, a przy tym konieczne są jedynie dwa przewody do transmisji (jednostronnej) sygnału. Niegdyś główną wadą tej techniki była niższa szybkość transmisji oraz bardziej skomplikowana budowa wewnętrzna interfejsu (konieczność istnienia rejestru przesuwnego). Lecz zalety tego typu przesyłania danych przesłoniły wady i dziś, po zwiększeniu szybkości łącz szeregowych głównie poprzez wzrost częstotliwości pracy interfejsów oraz stosowanie zaawansowanych metod kodowania sygnału, transmisja równoległa jest w zasadzie wykorzystywana jedynie we wnętrzu komputera, natomiast większość wyprowadzeń interfejsów na zewnątrz (USB, SATA, HDMI, DisplayPort, Thunderbolt i inne) realizowana jest jako szybkie łącza szeregowe.

Tryby pracy podczas transmisji

Fizyczna metoda transmisji to jedno, a sposób w jaki jest ona sterowana, w jaki sposób określa się które dane i kiedy mają zostać przesłane, i kto nadzoruje transmisję - to zupełnie inne, choć równie istotne zagadnienie. Istnieją trzy podstawowe sposoby realizacji operacji wejścia-wyjścia (ang. *input-output* - IO):

- **programowane wejście-wyjście** - komunikacja odbywa się pomiędzy procesorem a modułem I/O. Procesor za pośrednictwem specjalnego programu ma możliwość sterowania operacją we-wy, wysyłania rozkazu zapisu/odczytu i transferu danych. Wadą tego rozwiązania jest fakt, że procesor musi czekać na zakończenie operacji wejścia-wyjścia przez moduł I/O. Jeśli weźmiemy pod uwagę zdecydowanie większą szybkość współczesnych procesorów niż prędkość osiągana przez urządzenia zewnętrzne, to nie ma co się dziwić, że tego typu podejście nie jest współcześnie stosowane w dużych systemach ogólnego przeznaczenia do przesyłania dużych ilości danych, choć w prostych układach mikroprocesorowych i mikrokontrolerach nadal bywa używane;
- **wejście-wyjście sterowane przerwaniami** - procesor wydaje rozkaz I/O, później zajmuje się wykonywaniem innych rozkazów, gdy moduł I/O zakończy wykonywanie swojego zadania przerywa czynność procesora.

  W obu powyższych rozwiązaniach procesor odpowiada za odczyt danych z pamięci głównej i za zapis tych danych;
- **bezpośredni dostęp do pamięci** (ang. *direct memory access - DMA)*) - tryb ten charakteryzuje się tym, że moduł I/O i pamięć główna wymieniają dane bez udziału procesora.

Programowe wejście/wyjście

Urządzenia zewnętrzne są zwykle połączone z systemem poprzez moduły wejścia-wyjścia. Oczywiście każde urządzenie posiada unikatowy identyfikator i adres, który wystawia procesor chcąc odwołać się do urządzenia. Jeżeli procesor, pamięć główna i moduły wejścia-wyjścia używają wspólnej magistrali, możliwe są dwa tryby adresowania:

1. **Odwzorowany w pamięci** - ta sama przestrzeń adresowa przeznaczona jest dla komórek pamięci i urządzeń I/O. Procesor wykonuje takie same instrukcje maszynowe niezależnie od tego, czy odwołuje się do pamięci czy do urządzeń I/O. Tak więc, na magistrali potrzebna jest tylko jedna linia zapisu i jedna linia odczytu.
2. **Izolowane wejście-wyjście** - przestrzeń adresowa jest podzielona: inną pulę adresów mają urządzenia I/O, a inną komórki pamięci. Tak więc, część linii magistrali wykorzystywana jest do odczytu/zapisu komórek pamięci, a część do adresowania modułów I/O, dzięki czemu możliwe jest wykorzystanie całej przestrzeni adresowej zarówno przez pamięć jak i urządzenia I/O.

Wejście / wyjście sterowane przerwaniami

Wadą programowanego wejścia-wyjścia był długi czas oczekiwania przez procesor na gotowość wybranego modułu I/O. Dlatego też wykorzystuje się wejście-wyjście sterowane przerwaniami, którego działanie można przedstawić następująco:

1. Procesor wydaje modułowi rozkaz wejścia-wyjścia, po czym przechodzi do wykonywania "ciekawszych" zadań (np. do obliczania całki Fouriera ...).
2. Kiedy moduł I/O jest gotowy do wymiany danych z procesorem (czyli np. wczytał już dane z urządzenia peryferyjnego), wysyła żądanie obsługi przerwania i czeka aż procesor będzie gotowy do wymiany danych.
3. Gdy następuje przerwanie od modułu I/O procesor zachowuje kontekst bieżącego programu (np. obliczania całki Fouriera) czyli minimalną zawartość licznika programu i rejestrów procesora, a następnie zgłasza chęć transmisji.
4. Moduł I/O umieszcza dane na magistrali, po czym jest gotowy do kolejnej operacji wejścia-wyjścia.

5. Procesor transmituje dane, a następnie odnawia kontekst programu (np. całki Fouriera) i wznawia jego działanie.

Wejście-wyjście sterowane przerwaniami jest wydajniejsze niż programowane, jednakże i w tym rozwiązaniu traci się dużo czasu procesora, który musi nadzorować każde słowo transmisji pomiędzy pamięcią a modułami I/O. Każde słowo - co oznacza że procesor 32 bitowy ma "przerwę" jedynie na czas potrzebny na przygotowanie przez urządzenie zewnętrzne 4 bajtów ... co nie trwa długo. I co każde 4 bajty urządzenie przesyłające np. zdjęcie z aparatu cyfrowego "dzwoni" do procesora i chce pogadać ... przecież tak nie da się wydajnie pracować! ;) Dlatego też w przypadku przesyłania dużych bloków danych (np. przy kopiowaniu dużych plików czy transmisji strumieni wideo) wejście-wyjście sterowane przerwaniami jest nieefektywne i zastępowane przez DMA, natomiast wciąż pozostaje podstawową metodą obsługi wielu urządzeń generujących pojedyncze zdarzenia (klawiatura, mysz, interfejsy sieciowe i inne).

DMA

Metoda **bezpośredniego dostępu do pamięci** DMA (ang. *Direct Memory Access*) stosowana jest zawsze tam, gdzie muszą być przenoszone duże ilości danych przy minimalnym obciążeniu procesora. To "minimalne" obciążenie w praktyce oznacza zupełne zwolnienie procesora z nadzorowania przesyłania danych między pamięcią a urządzeniem zewnętrznym.

Metoda ta wymaga zainstalowania dodatkowego modułu DMA na magistrali systemowej. Moduł ten potrafi przejmować od procesora sterowanie magistralą systemową. Transfer ciągle inicjuje procesor, ale w tym wypadku jest to raczej polecenie szefa do dobrze wykształconego zespołu specjalistów. Gdy procesor chce odczytać/zapisać dane wydaje rozkaz modułowi DMA, wysyłając następujące informacje:

- czy potrzebny jest odczyt, czy zapis?
- adres urządzenia I/O wymaganego do komunikacji;
- adres początkowej komórki pamięci, do której nastąpi zapis/odczyt;
- liczbę słów, które należy odczytać/zapisać.

Procesor powraca do wykonywania swoich zadań (np. programu użytkowego), zaś moduł DMA zajmuje się transmisją danych pomiędzy modułem I/O a pamięcią z pominięciem procesora. Gdy moduł DMA zakończy przesył danych, wysyła sygnał przerwania do procesora. Procesor jest więc używany tylko na początku i końcu transmisji - niezależnie od ilości przesyłanych danych.

Jak już wspomnieliśmy moduł DMA przejmuje działanie procesora w komunikacji pomiędzy pamięcią a modułami I/O. Dlatego też konieczny jest dostęp do magistrali. I tu pojawiają się dwie opcje:

1. Moduł DMA używa magistrali tylko wówczas, gdy nie potrzebuje jej procesor.
2. Moduł DMA wymusza czasowe zawieszenie operacji procesora.

Ta ostatnia metoda jest bardziej popularna i określa się ją mianem **wykradania cyklu** ze względu na fakt, że moduł DMA zajmuje cykl magistrali.

Wspomniane "wykradanie cyklu" nie jest równoważne przerwaniu - procesor zatrzymuje się na jeden cykl magistrali bez zachowywania kontekstu. Oczywiście "wykradanie cyklu" sprawia, że procesor pracuje nieco wolniej, ale dla dużych bloków danych metoda DMA jest daleko bardziej efektywna niż opisane wcześniej *programowane wejście-wyjście* i *wejście-wyjście sterowane przerwaniami*.

1.9. Mechanizmy ochrony wejścia/wyjścia, pamięci, jednostki centralnej

Ochrona wejścia/wyjścia

Podobnie jak w przypadku omawiania przerwań, również w tym segmencie zajmiemy się problemem, którego rozwiązanie wymaga współdziałania elementów sprzętowych i systemu operacyjnego. Przerywania są nam potrzebne, by można było chwilowo wstrzymać wykonywanie jednego programu i przesłać dane z urządzeń wejścia/wyjścia lub przekazać sterowanie tymczasowo do innego programu. To pozwala na stosowanie we współczesnych systemach operacyjnych wielozadaniowości - mamy jeden procesor, który jednocześnie wykonuje wiele programów (w rzeczywistości w danej chwili wykonuje zawsze jeden, ale bardzo szybko przełącza się między nimi). Jednak, aby takie działanie było możliwe na szeroką skalę, samo przydzielanie czasu procesora nie wystarczy. Trzeba jeszcze zadbać, by jeden niedbale napisany program nie uszkodził pozostałych, pracujących z nim równolegle.

Działania systemu mogą zostać zakłócone poprzez nieautoryzowany dostęp do pamięci: program A korzysta z pamięci programu B zamazując jego dane. Mogą być również zakłócone poprzez jednoczesny dostęp do tego samego urządzenia wejścia/wyjścia - zdają sobie dobrze z tego sprawę osoby, które pisały programy w asemblerze ... pozostali niech na razie uwierzą nam na słowo ;)

Niedozwolone operacje I/O prowadzą do nieokreślonych zachowań systemu komputerowego. Dlatego też, aby ustrzec się przed wykonaniem niedozwolonej operacji wejścia-wyjścia przyjęto, że wszystkie rozkazy I/O są uprzywilejowane i mogą być wykonywane jedynie w trybie jądra systemu operacyjnego. Program użytkownika może korzystać z urządzeń wyłącznie pośrednio — poprzez wywołania systemowe kierowane do sterowników urządzeń pracujących w przestrzeni jądra. Oprócz tego współczesne procesory wykorzystują jednostkę MMU oraz mechanizm stronicowania (paging), aby dodatkowo kontrolować dostęp procesów do przestrzeni adresowej, uniemożliwiając im bezpośredni dostęp do obszarów pamięci, w których znajdują się struktury systemowe odpowiedzialne za operacje I/O.

Współczesne procesory pracują co najmniej w dwóch trybach: trybie użytkownika (ang. *user mode*) oraz trybie jądra (ang. *kernel mode*). W trybie użytkownika uruchamiane są zwykłe programy — przeglądarki, gry, aplikacje biurowe — które mają jedynie ograniczony dostęp do zasobów komputera. Nie mogą wykonywać uprzywilejowanych instrukcji ani odwoływać się bezpośrednio do sprzętu.

W trybie jądra działa system operacyjny, jego sterowniki oraz procedury obsługi przerwań. Kod działający w tym trybie ma pełne uprawnienia: może wykonywać rozkazy I/O, zarządzać pamięcią, konfigurować urządzenia oraz sterować przełączaniem procesów. Dzięki temu system operacyjny pełni rolę „pośrednika” — zapewniając aplikacjom dostęp do zasobów sprzętowych, ale jednocześnie uniemożliwiając im wykonanie operacji, które mogłyby zakłócić pracę innych programów lub całego systemu.

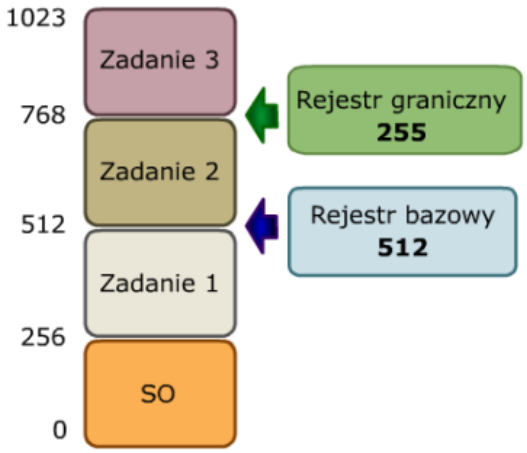
Przejście z trybu użytkownika do trybu jądra odbywa się wyłącznie za pomocą specjalnych instrukcji procesora, zwanych pułapkami lub wywołaniami systemowymi (ang. *system calls*). To właśnie poprzez takie wywołania program może poprosić system operacyjny o wykonanie zadania uprzywilejowanego — na przykład odczyt pliku, komunikację z kartą sieciową lub przydział dodatkowej pamięci. Po zakończeniu operacji system powraca do trybu użytkownika.

Rozdzielenie wykonania na dwa tryby jest kluczowym elementem bezpieczeństwa. Gdyby wszystkie programy działały w jednym trybie, każdy z nich mógłby dowolnie modyfikować pamięć, przejmować urządzenia I/O lub blokować procesor w nieskończonej pętli. Architektura trybów pracy procesora — wraz z ochroną pamięci realizowaną przez MMU — stanowi fundament, który umożliwia wielozadaniowość, stabilność i odporność systemów operacyjnych na błędy i ataki.

Ochrona pamięci

Podstawowymi jednostkami pamięci, które muszą być chronione, są wektor przerwań oraz systemowe procedury obsługi przerwań. Zaniedbanie tej ochrony mogłoby doprowadzić do sytuacji, w której program użytkownika zastąpiłby systemowe procedury obsługi przerwań skokami do własnego obszaru, przechwytując w ten sposób sterowanie od procedury obsługi przerwania. To doprowadzić mogłoby do nieprawidłowego działania systemu komputerowego ... co zresztą skwapliwie wykorzystywały wczesne wersje wirusów komputerowych.

Jedną z klasycznych metod ochrony pamięci programu jest wykorzystanie dwóch rejestrów: bazowego (ang. *base*) i granicznego (ang. *limit*). Mechanizm ten był powszechnie stosowany w starszych architekturach oraz w wielu mikrokontrolerach. W nowoczesnych komputerach osobistych stosuje się zwykle bardziej zaawansowaną ochronę pamięci opartą na stronicowaniu (paging) i jednostce MMU, dzięki czemu każdy proces otrzymuje własną wirtualną przestrzeń adresową izolowaną od pozostałych.



Rejestr bazowy i graniczny

Dzięki ww. rejestrom można zdefiniować zakres adresów (obszar czy też fragment pamięci) przyznawany niezależnie i w sposób nie pokrywający się - dla każdego z uruchomionych programów, oraz uzyskujemy możliwość ochrony pamięci poza tymi adresami - to czyni każdy sensowny system operacyjny.

Ochrona w skrócie polega na tym, iż procesor zanim zapisze / odczyta coś z pamięci, sprawdza dla każdego adresu, dla którego zleci takie działanie system operacyjny, jaka jest zawartość opisanych powyżej rejestrów. Jakakolwiek ingerencja wykonana przez program do pamięci innego programu lub innego, nie przydzielonego obszaru RAM, kończy się wygenerowaniem przez system błędu ochrony pamięci, który w Windows nosi nazwę *Access Violation*, a w systemach Unix/Linux odpowiada mu sygnał *SIGSEGV* (segmentation fault).

Ochrona jednostki centralnej

System operacyjny powinien sprawować stałą kontrolę nad komputerem, dlatego też należy zapobiec sytuacji, w której program użytkownika wpadłby w nieskończoną pętlę permanentnie zajmując procesor. Wykorzystuje się do tego **czasomierz** (ang. *timer*). Timer pozwala na generowanie przerwania po wyznaczonym czasie - czas ten może być stały lub zmienny (przyrostowo o pewien krok).

Tak więc system operacyjny, przed oddaniem sterowania do programu użytkownika, ustawia timer na przerwanie. Gdy czasomierz wywołuje przerwanie, sterowanie wraca do systemu operacyjnego, który decyduje: czy wystąpił błąd, czy też należy dać programowi użytkownika więcej czasu. W ten łatwy sposób można zapobiec zbyt długiemu działaniu programów lub wymusić zakończenie działania programu po upływie określonego czasu.

Czasomierz wykorzystywany jest również do podziału czasu procesora. To właśnie jego cykliczne przerwania umożliwiają tzw. *preemptive multitasking*, czyli wymuszane przez system operacyjny przełączanie się pomiędzy programami co pewien kwant czasu.

Omówione w tym segmencie mechanizmy koncentrują się głownie na sprzętowych aspektach bezpiecznej współpracy wielu programów oraz systemu operacyjnego. Na razie jeszcze nie skupialiśmy się na tym, *jak* przydzielać pamięć dla wielu programów, *ile* czasu przeznaczyć dla każdego z nich, *kiedy* przyznać dostęp do urządzeń zewnętrznych a kiedy go odmówić. Postaramy się odrobinę wprowadzić Was w te zagadnienia w lekcjach 5 i 6, lecz to także nie wyczerpie tematu. Część z Was będzie miała okazję później, w dalszym toku studiów, poznać wspomniane zagadnienia dokładniej, na przedmiocie **Systemy operacyjne**.

Spis treści

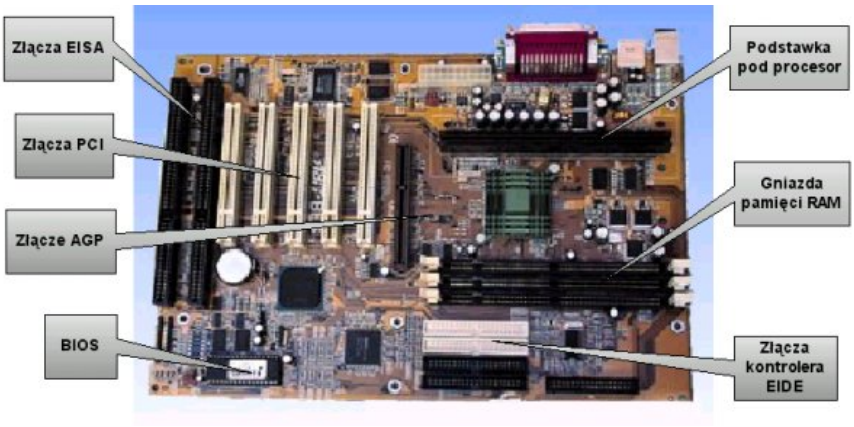
1. Lekcja 3
1.1. Procesory
1.2. Pamięć operacyjna
1.3. Karty graficzne
1.4. Pamięci masowe
1.5. Urządzenia wejściowe
1.6. Urządzenia wyjściowe
1.7. Interfejsy

1. Lekcja 3

Płyta główna

Omawianie współczesnych podzespołów komputera rozpoczniemy od jego szkieletu - czyli płyty głównej.

Płyta główna jest płytką z obwodami drukowanymi, na której montowana jest znaczna część komponentów niezbędnych do poprawnego funkcjonowania systemu komputerowego. Na płycie głównej umieszczona jest podstawka umożliwiająca montaż procesora, tutaj znajdują się również magistrale systemowe, pamięć ROM (a w praktyce najczęściej pamięć flash) służąca do przechowywania głównego programu obsługi komputera – klasycznego BIOS-u lub jego współczesnego następcy UEFI – a także programów testujących podzespoły komputera przy starcie systemu (POST), gniazda (ang *slot*) na karty rozszerzeń, gniazda pamięci RAM, kontroler urządzeń I/O, kontroler DMA, wszelkiego rodzaju wyprowadzenia interfejsów służących do podłączania dysków i urządzeń peryferyjnych, i - *last but not least* - chipset płyty głównej.



Budowa płyty głównej

Chipset

Chipset stanowi "serce" płyty głównej i odpowiada za sterowanie przepływem strumienia danych (sterowanie magistralą systemową). Jest to taka "szara eminencja". Niby rządzi procesor, ale bez zgody chipsetu, żadna jego decyzja nie zostanie dostarczona do urządzeń zewnętrznych.

Klasycznie chipset był podzielony logicznie na dwa osobne układy, tzw. mostki. **Mostek północny** (ang. *north bridge*) sterował przepływem danych pomiędzy procesorem, pamięcią operacyjną i najszybszymi magistralami (np. AGP, pierwsze PCI Express), natomiast **mostek południowy** (ang. *south bridge*) obsługiwał wolniejsze interfejsy wejścia/wyjścia (porty I/O, magistrale PCI/USB, kontrolery dysków itp.). W nowoczesnych komputerach wiele funkcji dawnego mostka północnego – w szczególności kontroler pamięci i linie PCIe – zostało przeniesionych bezpośrednio do procesora, a pozostałe funkcje mostków integrowane są w jednym układzie (np. Platform Controller Hub – PCH firmy Intel). Dzięki temu uproszczono konstrukcję płyty głównej i zwiększono wydajność komunikacji między CPU a pamięcią i urządzeniami zewnętrznymi. [Źródło: Platform Controller Hub]

Mostek południowy (ang. *south bridge*) umożliwia dołączenie do procesora portów I/O (tj. interfejsy szeregowe/równoległe, magistrala USB), zapewnia możliwość korzystania z magistrali ISA oraz pozwala podłączyć urządzenia do złącz IDE. Ponadto mostek południowy steruje funkcjami zarządzania energii oraz monitoruje parametry systemu.

Mostek północny (ang. *north bridge*), który steruje przepływem danych (jest kontrolerem FSB - głównej szyny procesora) pomiędzy procesorem, pamięcią operacyjną i podręczną, złączem AGP i PCI, a także mostkiem południowym. Mostek północny zapewnia bezkonfliktową współpracę magistrali pracujących nierzadko z różnymi częstotliwościami taktowania.

Pojęcie chipsetu ściśle wiąże się z typem obsługiwanego przez płytę główną procesora - innego chipsetu wymaga procesor Intel Core a innego AMD. Rodzaj chipsetu zależy również od specyfikacji elektrycznej i mechanicznej wyprowadzeń gniazda procesora (a więc typu gniazda). Dlatego przy budowaniu własnego zestawu komputerowego w pierwszej kolejności powinniśmy wybrać procesor, w oparciu o który będziemy składać resztę podzespołów. Typ gniazda procesora obecny na płycie głównej musi być zgodny z konkretnym modelem układu. Do najpopularniejszych zaliczamy:

Procesory Intel:

- LGA 775 – układy Pentium 4, Celeron, Core 2 Duo
- LGA 1156 – układy Intel Core i3, i5, i7 oraz Core 2 Duo
- LGA 1155 – procesory oparte o architektury Sandy Bridge i Ivy Bridge
- LGA 2011 – procesory oparte o architektury Sandy Bridge-E i Ivy Bridge-E
- LGA 1150 – procesory oparte o architektury Haswell i Broadwell

Procesory AMD:

- Socket AM2+ – układy Phenom X4, Phenom II, wsparcie dual-channel DDR3-SDRAM i HyperTransport 3
- Socket AM3 – układy Phenom X6, Athlon X4, wsparcie dual-channel DDR3-SDRAM i HyperTransport 3
- Socket AM3+ – procesory oparte o architekturę Bulldozer
- Socket FM2 – procesory oparte o architekturę Trinity

Powyższe przykłady dotyczą popularnych gniazd stosowanych w kilku generacjach procesorów Intel Core i AMD Phenom/Athlon/FX oraz pierwszych APU. W późniejszych latach pojawiły się kolejne sockety, m.in. LGA 1151, LGA 1200 i LGA 1700 dla nowszych generacji procesorów Intel Core oraz AM4 i AM5 dla nowoczesnych procesorów AMD Ryzen. Dobierając płytę główną, zawsze trzeba sprawdzić zgodność gniazda z konkretną generacją procesora, a nie tylko z producentem. [Źródła: Intel Core – gniazda, CPU socket]

Sam chipset może być wyprodukowany przez kilka firm, najbardziej znane to Intel, AMD, NVidia czy SIS. Dokładne omówienie istniejących chiset-ów to temat - rzeka, przy czym, podobnie jak w przypadku procesorów - rozwiązania w tej dziedzinie zmieniają się wyjątkowo szybko. Zamiast więc zaprzątać Wam głowę szczegółami, chcielibyśmy, byście pamiętali o znaczeniu płyty głównej. To od jakości płyty właśnie w dużym stopniu zależy stabilność pracy komputera oraz jego możliwości rozbudowy. Dlatego też nie polecamy stosowania rozwiązań najtańszych (aktualnie są to płyty kosztujące ok. 100-300 PLN). Jeśli macie wpływ na wybór płyty głównej do komputera - miejcie świadomość że większość z tych najtańszych ma problemy z synchronizacją, obsługą pamięci z pełną szybkością, oraz pracą z najszybszymi procesorami i kartami rozszerzeń. Na drugim krańcu są rozwiązania przeznaczone dla tzw. overclock-erów - czyli osoby które samodzielnie podnoszą częstotliwości taktowania oraz dobierają napięcia zasilające poszczególne elementy systemu komputerowego (głównie procesor i pamięć). Takie płyty kosztują zazwyczaj ponad 1000 PLN, są wykonane z najwyższą jakością, jednak przeciętny użytkownik komputera nigdy nie wykorzysta ich wszystkich możliwości. Do większości zastosowań najlepsze jest rozwiązanie pośrednie - firmowe płyty główne, solidnie wykonane lecz bez wodotrysków. Wydając trochę więcej niż na najtańsze rozwiązanie - kupujemy sobie spokój... ;)

Istotne jest dopasowanie płyty do rozmiaru obudowy. Aktualnie na rynku w tej dziedzinie obowiązuje kilka standardów:

- ATX – wymiary płyty to 305 x 244 mm
- microATX – wymiary płyty to 244 x 244 mm
- miniITX – wymiary płyty to 170 x 170 mm
- nanoITX – wymiary płyty to 120 x 120 mm
- picoITX – wymiary płyty to 100 x 72 mm

W typowych komputerach stacjonarnych najczęściej spotyka się dziś płyty w formatach ATX, microATX i mini-ITX, natomiast nano-ITX i pico-ITX są używane głównie w małych, wyspecjalizowanych systemach wbudowanych. [Źródło: Motherboard form factor]

Ponadto ważne jest, by płyta była wyposażona we wszystkie niezbędne interfejsy.

Ważniejsze wyprowadzenia

ISA (ang. *Industry Standard Architecture*)

Najstarszy, i w zasadzie już wychodzący z użycia standard 16-bitowej magistrali danych, umożliwiającej dołączanie dodatkowych kart rozszerzeń i oferującej "imponującą" przepustowość 8,33 MB/s. Rozwiązanie nie występuje już we współczesnych płytach głównych.

EISA (ang. *Extended Industry Standard Architecture*)

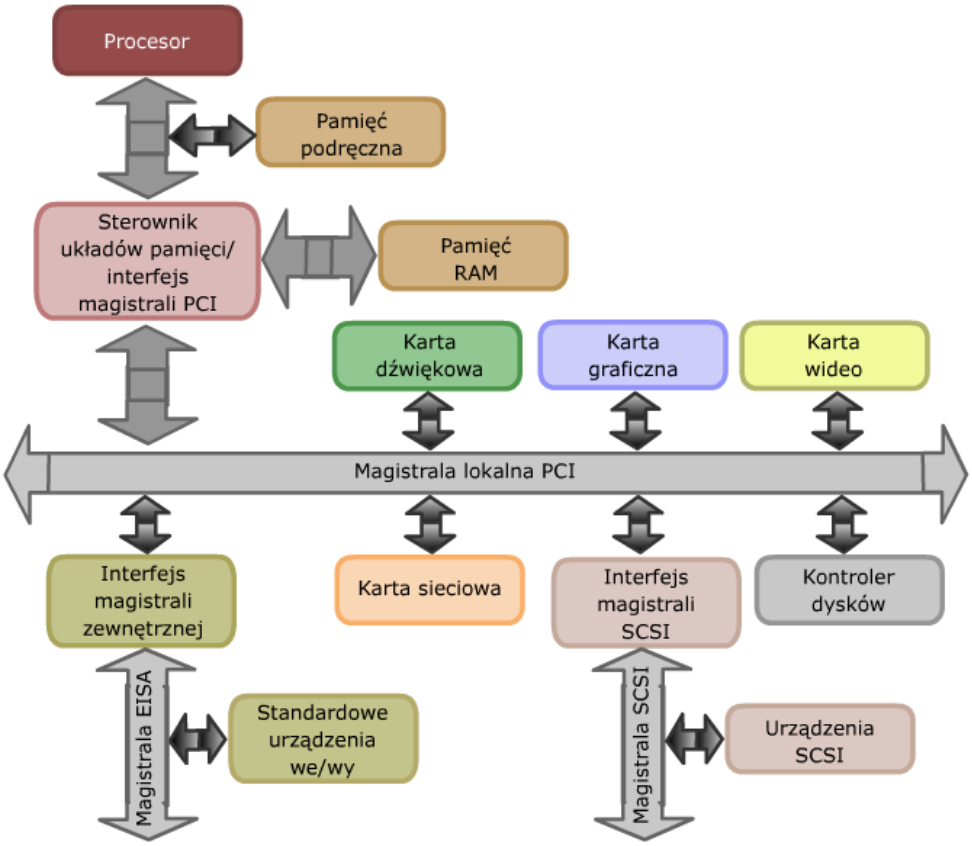
Zgodna ze swoją 16-bitową poprzedniczką, 32-bitowa magistrala zewnętrzna, która ze względu na kompatybilność wstecz, pracuje z prędkością 8,33 MHz, jednak dostęp do pamięci odbywa się z pełną szybkością 33 MB/s. Ze względu na niezbyt spektakularne osiągi i dużą cenę wytwarzania, magistrala ta jest coraz rzadziej wykorzystywana. Rozwiązanie nie występuje już we współczesnych płytach głównych.

VESA Local Bus (ang. *Video Electronics Standards Association*)

Standard ten najbardziej popularny w latach 1993-1996 miał stanowić wsparcie dla szybkich kart graficznych. Stanowił swoiste rozszerzenie technologii ISA o dodatkową magistralę danych, zwiększającą szybkość transferu pomiędzy procesorem a kartą graficzną, teoretycznie do 120 MB/s. 32-bitowa szyna VESA taktowana była zegarem procesora, którego częstotliwość pracy nie mogła przekraczać 40 MHz [1]. W pewnym czasie technologia stanowiła konkurencję dla droższej EISA, jednak szybko została wyparta przez technologię PCI. Rozwiązanie nie występuje już we współczesnych płytach głównych.

PCI (ang. *Peripheral Component Interconect*)

Architektura ta została wprowadzona w 1993 roku przez firmę Intel i przez wiele lat była najpopularniejszym rodzajem gniazd rozszerzeń w komputerach osobistych. Współcześnie jej rolę w nowych komputerach przejęła magistrala PCI Express, a złącza PCI spotyka się głównie w starszych lub specjalistycznych konstrukcjach. Zaletą magistrali PCI jest możliwość tworzenia złożonych systemów (specyfikacja 2.1 PCI opisuje, że możliwa jest współpraca do 256 magistrali, z których każda może obsługiwać do 32 urządzeń PCI). Ciekawostką świadczącą o możliwościach eskalacji PCI niech będzie fakt, że w typowym PC wykorzystywana jest tylko jedna magistrala PCI obsługująca do 10 urządzeń [1]. Magistrala PCI może pracować z szybkością od 0 do 33 MHz (ver. 2.1 do 66 MHz), co daje przepustowość 132 MB/s. Magistrala pozwala na dostęp do adresowanego obiektu w jednym takcie zegara. Zdefiniowane jest również 64-bitowe złącze magistrali PCI, pracujące z prędkością do 264 MB/s.
Architekturę komputera ze złączem PCI przedstawia poniższy rysunek:



Architektura komputera z magistralą PCI

AGP (ang. *Accelerated Graphics Port*)

AGP stanowi pewne przedłużenie magistrali PCI, dlatego określana jest jako interfejs komunikacyjny. Magistrala AGP nie przyspiesza operacji graficznych, a jedynie umożliwia bardziej wydajną pracę procesora graficznego [1]. Dzieje się tak, gdyż ów procesor sprawuje wyłączną kontrolę nad magistralą. AGP bazuje na specyfikacji PCI 2.1, zmienia jednak znaczenie niektórych sygnałów i wprowadza szereg nowych. Magistrala AGP może pracować w jednym z trzech trybów:

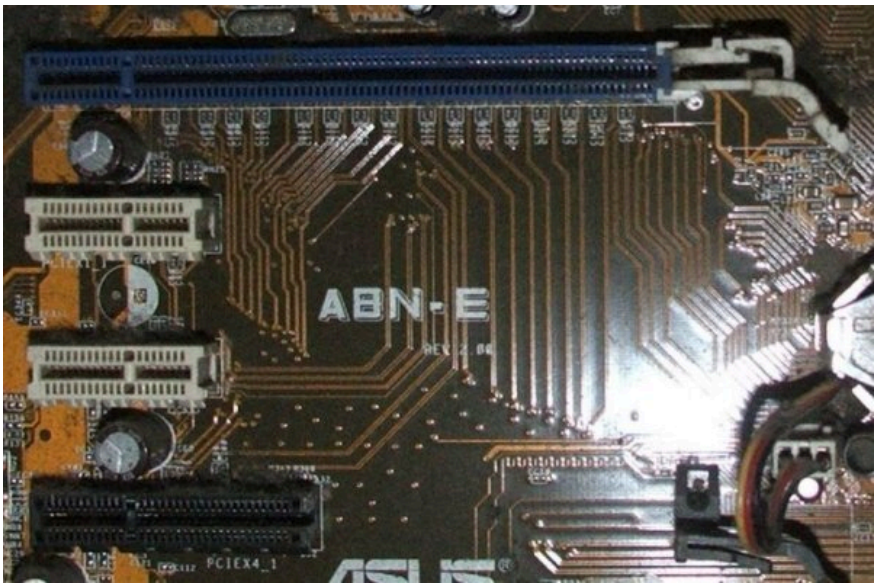
- Tryb 1x. Rozszerzenie standardu PCI, w którym dzięki podwojeniu częstotliwości pracy zegara taktującego do 66 MHz, uzyskano teoretyczny maksymalny transfer 264 MB/s.
- Tryb 2x. Częstotliwość zegara jest taka jak poprzednio, ale wymiana danych odbywa się podczas narastającego i opadającego zbocza sygnału taktującego. Daje to wzrost częstotliwości zegara do 133 MHz , a teoretyczna przepustowość wynosi 532 MB/s.
- Tryb 4x. Różnica w stosunku do trybu poprzedniego polega na tym, że w czasie zbocza wykonywane są dwie transmisje, a więc teoretyczna przepustowość wzrasta do 1064 MB/s. Tryb ten pracuje na znacznie obniżonym poziomie napięć - 1.5V.

Standard AGP był przez wiele lat podstawowym złączem dla kart graficznych, jednak we współczesnych komputerach został całkowicie zastąpiony przez magistralę PCI Express, która oferuje wyższą przepustowość i większą elastyczność konfiguracji. [Źródło: PCI Express]

PCI Express

Magistrala PCI Express (PCI-s PCIe, PCI-E), znana również jako 3GIO (od 3rd Generation I/O), jest szeregową magistralą typu punkt–punkt, służącą do podłączania urządzeń do płyty głównej. Ma ona zastąpić magistralę PCI oraz AGP. PCI-Express stanowi magistralę lokalną typu szeregowego, łączącą dwa punkty (Point-to-Point). Nie jest to więc magistrala w tradycyjnym rozumieniu, i nie jest rozwinięciem koncepcji "zwykłego" PCI. Taka konstrukcja eliminuje konieczność dzielenia pasma pomiędzy kilka urządzeń - każde urządzenie PCI-Express jest połączone bezpośrednio z kontrolerem. Sygnał przekazywany jest za pomocą dwóch linii, po jednej w każdym kierunku. Częstotliwość taktowania wynosi 2.5GHz. Protokół transmisji wprowadza dwa dodatkowe bity, do każdych ośmiu bitów danych. Zatem przepustowość jednej linii wynosi 250MB/s. W związku z tym, że urządzenia mogą jednocześnie przekazywać sygnał w obydwu kierunkach (full-duplex) to można ewentualnie przyjąć, że w przypadku takiego wykorzystania złącza, transfer może sięgać 500MB/s. Możliwe jest kilka wariantów tej magistrali - z 1, 2, 4, 8, 12, 16, 24 lub 32 liniami (każda składająca się z dwóch 2 pinowych części - nadawczej i odbiorczej). Wraz ze wzrostem liczby linii wydłużeniu ulega gniazdo, jego konstrukcja (poprzez wspólną część początkową i jedynie dodawanie na końcu nowych linii) umożliwia włożenie wolniejszej karty do szybszego gniazda (w drugą stronę jest niemożliwe). Gniazdo 1x ma 18 pinów z każdej strony, gniazdo 4x - 32, gniazdo 8x - 49, zaś gniazdo 16x - 82 piny z każdej strony.

Opisane powyżej parametry dotyczą pierwszej generacji PCI Express (PCIe 1.0). Późniejsze wersje specyfikacji (PCIe 2.0, 3.0, 4.0, 5.0 i nowsze) sukcesywnie podwajały przepustowość na linię, m.in. poprzez zwiększanie szybkości sygnału oraz zmianę sposobu kodowania (z 8b/10b na 128b/130b). W efekcie współczesne złącza PCIe x16 oferują przepustowości rzędu kilkudziesięciu gigabajtów na sekundę i są standardem dla nowoczesnych kart graficznych oraz bardzo szybkich dysków SSD NVMe. [Źródła: Synopsys – PCIe basics, PCIe standards]



Gniazdo magistrali PCI Express

Na płytach głównych gniazda 16x montuje się zwykle w miejscu gniazda AGP na starszych płytach (ponieważ większość chipsetów z kontrolerem PCI Express nie zawierają kontrolera AGP, najczęściej obecność PCI-E eliminuje możliwość użycia kart graficznych ze złączem AGP, przykłady chipsetów obsługujących zarówno AGP jak i PCI-E to: Via PT880 Pro dla procesorów Intela i ULi M1695 + ULI M1567 dla procesorów AMD), pod nim gniazda 8x, 4x i 1x, najdalej zaś od procesora - gniazda PCI.

Specyfikacja określa też mniejsze rozmiarowo warianty kart: miniExpress cards, ExpressCards (następca PCMCIA) oraz AdvancedTCA (następca CompactPCI).

AMR, CNR i PTI

Wszystkie trzy gniazda zostały opracowane przez firmę Intel i ułatwiać mają instalowanie prostych urządzeń rozszerzających.

AMR (ang. *Audio modem Riser*) - pozwala na dołączenie modemu lub karty dźwiękowej.
CNR (ang. *Communication and Network*) - pozwala na rozbudowę o kartę sieciową 10/100 Mb/s.
PTI (ang. *Panel link TV-out Interface*) - pozwala na podłączenie kart TV.
Standardy AMR, CNR i PTI pojawiały się głównie na starszych płytach głównych i dziś praktycznie nie występują już w nowych konstrukcjach, gdyż ich funkcje przejęły zintegrowane układy na płycie oraz urządzenia podłączane przez USB i PCI Express.

Złącza pamięci operacyjnej i masowych

SATA II / III to złącza wykorzystywane do podpinania pamięci masowej, przede wszystkim dysków twardych i napędów optycznych. Zdecydowanie lepszą opcją jest SATA III, gdyż jego przepustowość wynosi 6 Gbit/s, czyli ponad 2 razy więcej od SATA II. W nowych komputerach systemowych rolę najwydajniejszych nośników coraz częściej pełnią jednak dyski SSD w standardzie NVMe, podłączane bezpośrednio do magistrali PCI Express, które oferują wielokrotnie wyższe prędkości od klasycznych dysków SATA. [Źródło: PCI Express] Dzięki temu interfejsowi podłączymy do komputera dysk zewnętrzny, przy szybkości transferu 3 Gbit/s. Maksymalna długość kabla eSATA może wynosić 2 m, a w przeciwieństwie do USB, port ten nie potrzebuje zasilania (wyjątkiem jest hybrydowy port eSATAp). Interfejs eSATA był popularny jako szybkie zewnętrzne złącze dla dysków, ale we współczesnych komputerach został w dużej mierze wyparty przez porty USB 3.x, USB-C i Thunderbolt. [Źródło: Form factor & interfejsy] W przypadku pamięci operacyjnej - warto wybierać modele płyt głównych, które mają co najmniej 4 sloty na kości, dzięki czemu będziemy mieli możliwość rozbudowy pamięci operacyjnej w przyszłości.

BIOS / EFI

BIOS z definicji to zapisany w pamięci stałej komputera zestaw podstawowych procedur stanowiących ogniwo spajające między systemem operacyjnym a zamontowanymi podzespołami. BIOS (lub jego współczesny następca UEFI) jest programem zapisanym w pamięci ROM/flash płyty głównej i innych kart rozszerzeń. Większość firmware'u obecnie jest zapisywana w pamięciach typu EEPROM/flash, dzięki czemu jest możliwa ich aktualizacja.

W większości przypadków nie będziecie mieli potrzeby modyfikacji ustawień BIOS. Warto jednak mieć świadomość, że narzędzie to pozwala na bardzo wiele. W podstawowych wersjach BIOS-a, w niedrogich płytach głównych, znajdują się funkcje dotyczące zarządzania energią, bootowania systemu i ustawień dotyczących daty oraz godziny. Wiele modeli pozwala jednak na monitorowanie tak ważkich parametrów jak temperaturę procesora, napięcie czy prędkość obrotową wentylatora. To jednak nie wszystko – im bardziej wyspecjalizowana płyta główna (np. dla graczy lub overclockerów), tym bardziej zaawansowane funkcje są dostępne.

BIOS to nie jedyna opcja; już od jakiegoś czasu na rynku jest dostępny EFI (UEFI) – następca BIOS-u w komputerach osobistych z interfejsem znacznie bardziej przyjaznym dla użytkownika. Pomysł EFI zrodził się w firmie Intel na potrzeby procesora serwerowego Itanium. Jedną z jego cech charakterystycznych jest możliwość pisania dla niego sterowników, które są niezależne od systemu operacyjnego. EFI ma własne minisystemy operacyjne (UEFI Shell), które obsługuje się poleceniami tekstowymi. Co ciekawe, EFI umożliwia obsługę dysków twardych większych od 2 TB, maksymalnie do 8192 EB (eksabajtów). Od 2006 r. firma Apple stosuje EFI na swoich maszynach, choć nie ma możliwości

wejścia jego w zaawansowane ustawienia. Z EFI/UEFI może się jednak wiązać kilka niedogodności – na przykład bardziej rozbudowana konfiguracja (Secure Boot, listy kluczy, wybór trybu rozruchu) oraz problemy z uruchamianiem bardzo starych systemów operacyjnych, które nie mają pełnego wsparcia dla UEFI (np. Windows XP, Vista czy dawne dystrybucje Linuksa). Współczesne systemy, takie jak aktualne wydania Linuksa czy Windows 10/11, są projektowane z myślą o pracy w trybie UEFI i wykorzystują jego dodatkowe możliwości, w tym obsługę dużych dysków, Secure Boot i zaawansowane mechanizmy konfiguracji. [Źródła: UEFI i Windows 11, UEFI w Linuksie]

1.1. Procesory

Dominujące rozwiązania

Współcześnie na rynku dominują dwie architektury procesorów. Jedna z nich, ogólnie lepiej znana, to procesory rodziny x86 i jej rozwinięcia x86-64. Ta architektura dominuje na rynku komputerów osobistych z systemem Windows, komputerów przenośnych oraz serwerów, w zasadzie nie mając tam znaczącej konkurencji. Spośród firm, które produkują procesory zgodne z tą architekturą należy wymienić Intel oraz AMD - jako najpoważniejszych graczy. Z drugiej strony - stale rosnący rynek systemów kieszonkowych - tabletów, smartphone-ów, i podobnych rozwiązań opiera się na innej architekturze - ARM, bardziej wyrafinowanej i z mniejszym "historycznym ogonem" niż x86. Tutaj także mamy wielu producentów procesorów, z których najbardziej znani to Qualcomm, Apple, Freescale, Samsung czy NVidia. Warto przy tym pamiętać, że od 2020 r. również laptopy firmy Apple (MacBooki) korzystają z własnych procesorów ARM (rodzina Apple M), co oznacza, że architektura ARM stała się ważnym graczem także w segmencie komputerów osobistych [Źródło: Apple silicon]. W niniejszym rozdziale w skrócie przedstawimy Wam informacje o obu architekturach, wraz z opisem ich typowych przedstawicieli, włączając w to rozwiązania współczesne - jednakże pamiętajcie - ten opis zestarzeje się wyjątkowo szybko, dlatego też warto zawsze uzupełnić swoją wiedzę korzystając z internetu.

Architektura x86

Architektura x86 po raz pierwszy pojawiła się wraz z modelem programowym procesorów wykorzystanym przez Intela w 1978 roku w 16-bitowej procesorze Intel 8086. Procesor ten w 1981 roku wykorzystany został przez firmę IBM – a właściwie nieco jego okrojona wersja Intel 8088, w której zmniejszono do 8 bitów szerokość magistrali danych – do do produkcji protoplasty dzisiejszych komputerów osobistych, pierwszego komputera IBM PC. Popularność jaką zyskały komputery typu PC, a później ich następcy doprowadziła do popularyzacji architektury x86 i jej dalszego rozwoju, który trwa do dzisiaj, finalnie wypierając alternatywne, nowocześniejsze rozwiązania. Swoją nazwę architektura x86 zawdzięcza generacjom układów Intela, wykorzystujących zastosowany w procesorze Intel 8086 model programowy, a które oznaczane były kolejno jako Intel 80286, Intel 80386 oraz Intel i486. Procesor Intel i486 był też ostatnim modelem zawierającym na końcu nazwy cyfry 86. Jego następca, procesor piątej generacji zgodnej z modelem programowym x86, nosił nazwę Pentium.

Pierwsze układy x86 były typowymi realizacjami CISC - przetwarzającymi złożone rozkazy w kilku / kilkunastu cyklach zegara, co skutkowało dość skomplikowaną realizacją fizyczną procesora z jednej strony - z drugiej upraszczało jego programowanie. Co ważne, ten podstawowy model programowy architektury x86 przetrwał do dzisiaj – został on oczywiście przez lata rozbudowany i zmodyfikowany, ale kompatybilność w dół została zachowana. Dzięki temu nawet dziś na najnowszych układach zgodnych z architekturą x86 można bez problemu uruchomić najstarsze aplikacje pochodzące jeszcze z 1981 roku. Nie umożliwia tego w prosty sposób żadna inna platforma.

Pierwsze zmiany w architekturze x86 wprowadzone zostały już w drugim z tej rodziny układzie 80286, gdzie dodano nowe instrukcje, sposób adresowania pamięci, a także wprowadzono potokowe przetwarzanie danych. W 1985 roku pojawiła się 32-bitowa kość Intel 80386, znana też pod nazwą i386DX. W układzie tym wprowadzone zmiany były znacznie istotniejsze. Po pierwsze - zastosowano 32-bitową magistralę adresową oraz 32-bitową magistralę danych. Oczywiście rozszerzone zostały do 32-bitów też rejestry ogólnego przeznaczenia, co w połączeniu ze zmianami w jednostkach wykonawczych spowodowało, że był to procesor 32 bitowy. Dodano także nowe rejestry kontrolne. Istotną zmianą było dodanie do procesora jednostki zarządzania pamięcią MMU (Memory Management Unit), która umożliwiła nowocześniejsze mechanizmy ochrony pamięci i wielozadaniowości, wykorzystywane później przez systemy operacyjne z rodziny Windows NT czy Unix/Linux [Źródło: Intel 80386].

Następcą układów Intel 80386 stała się rodzina procesorów Intel i486 (na rynku pojawiły się 1989 rok), które pod względem modelu programistycznego były niemal identyczne z układami i386DX – dodano w nim jedynie kolejne instrukcje. Spore zmiany nastąpiły natomiast w mikroarchitekturze układu – po raz pierwszy pojawiła się wspólna dla danych i instrukcji pamięć podręczna pierwszego poziomu (cache L1) o pojemności 8 KB oraz zintegrowany z procesorem koprocesor arytmetyczny FPU (Floating Point Unit), wykonujący operacje zmiennoprzecinkowe. Co ważne, procesory i486 jako pierwsze spośród układów x86 większość instrukcji wykonywały dokładnie w jednym takcie zegara.

Kolejną, piątą generację procesorów zgodną z architekturą x86 nazwano Pentium - od greckiej liczby pięć (pente). Pierwszy model, który pojawił się w 1993 roku składał się, mówiąc w pewnym uproszczeniu, z połączonych ze sobą dwóch układów i486. Taka konstrukcja pozwalała, w zależności od sytuacji, wykonywać dwa rozkazy równolegle. Pewnie niewielu użytkowników komputerów osobistych pamięta procesory Pentium 60 MHz ... Te pierwsze procesory zasilane były napięciem 5V. Kolejna grupa procesorów Pentium (P54C) pracowała przy mniejszym napięciu zasilającym (3,3V), a częstotliwość taktowania jądra oscylowała w granicach 200 MHz.

Procesory Pentium są 32-bitowe - rejestry procesora mają 32-bity, jednakże wewnętrzne ścieżki mają nawet do 256 bitów szerokości, dzięki czemu prędkość wewnętrznych transferów jest większa niż mogłoby wynikać z owych 32-bitów. Pentium posiada dwa potoki przetwarzające instrukcje stałoprzecinkowe (U i V) oraz jednostkę zmiennoprzecinkową. Jeżeli jest to możliwe, procesor przetwarza w każdym cyklu dwie instrukcje (po jednej na potok). Potoki pracują wyłącznie w trybie synchronicznym, niemożliwe jest więc np. przewidywanie skoków przez jeden z nich, ponieważ zatrzymanie jednego z potoków prowadzi do zatrzymania drugiego. Jednak Pentium posiada dodatkowy moduł dynamicznego przewidywania rozgałęzień BPU (ang. *Branch Prediction Unit*). Moduł ten przewiduje rozgałęzienia, czyli "wkłada" do potoku te instrukcje, które jego zdaniem będą wykonywalne. Błędne przewidzenie rozgałęzienia powoduje, że jeden z potoków musi wstrzymać przetwarzanie, a więc i drugi potok nie może być realizowany.

Pentium może działać w systemach wieloprocesorowych, ale tylko z identycznym procesorem (*Dual system*). Montowany jest w gniazdach rozszerzeń typu Socket 7.

Istotna modyfikacja architektury x86 nastąpiła zaś w 1995 roku wraz z premierą układu Pentium MMX (oznaczenie kodowe P55C), kiedy to do listy rozkazów dodano zestaw 57 nowych instrukcji MMX (MultiMedia eXtension) oraz specjalny moduł wykonawczy do ich przetwarzania. Procesor Pentium MMX był jednostką typu SIMD (Single Instruction Multiple Data), czyli wykonuje operację na kilku grupach danych jednocześnie. Rozszerzenie spowodowało pojawienie się nowych instrukcji (i nowych wyprowadzeń) oraz nowych rejestrów wykorzystywanych do tego celu. Istotne zmiany procesora Pentium MMX w stosunku do jego poprzednika, to:

- podwójne napięcie zasilające (od wersji 166 MHz) - układy odpowiadające za współpracę z magistralami I/O zasilane są innym napięciem niż rdzeń procesora;
- zwiększona pamięć podręczna L1 (ang. *first-level cache*) do 16kB - pracuje z poczwórną asocjacją;
- stos powrotu (ang. *return stack*) - przy przejściu do wykonywania podprogramu zapamiętywany jest adres powrotu do programu aktualnie wykonywanego;
- bufor zapisu jest zwiększony z 2 do 4 słów;
- nowa doskonalsza jednostka BPU (taka jak w Pentium Pro);
- możliwość równoległego wykonywania rozkazów (w sprzyjających warunkach 2 polecenia równolegle);

Modele Pentium MMX dzięki wykorzystaniu ww. nowinek (ale bez nowych instrukcji) osiągnęły wzrost mocy obliczeniowej do ok. 20 %.

Pentium MMX był ostatnim procesorem Intela wykorzystującym mikroarchitekturę CISC. Kolejne generacje układów zgodnych z x86 tego producenta tylko zewnętrznie są układami CISC, a wewnątrz są już klasycznymi układami typu RISC. We wszystkich nowoczesnych procesorach rozkazy x86 przetwarza się bowiem na proste mikropolecenia zgodne z architekturą RISC. Co ciekawe, pierwszym układem zgodnym z architekturą x86, który wewnętrznie był procesorem RISC był NextGen Nx586 z 1994 roku, na bazie którego powstał w 1995 roku AMD K5.

Pierwszym układem Intela z wewnętrzną architekturą RISC był Pentium Pro, również z 1995 roku, który przeznaczony był głównie do serwerów i wydajnych stacji roboczych. Na jego bazie powstawały kolejno układy Pentium II (w 1997 roku) i Pentium III (1999), a później Intel Core (2006), Core 2 Duo (2006), a także współczesne Intel Core i3, i5 oraz i7. Przy czym wraz z kolejnymi generacjami procesorów pojawiały się kolejne modyfikacje (czy też raczej rozszerzenia) modelu programistycznego x86, np pojawienie się się rozkazów i jednostek SSE (Streaming SIMD Extension) w procesorze Pentium III, które są znacząco wydajniejsze niż MMX.

Pentium II to model, który pierwszy przełamał granicę taktowania 200 MHz (stąd już tylko krok do 1GHz). Tym co odróżniło go od poprzedników był zupełnie nowy image. Otóż procesor Pentium II pojawił się na pokładzie specjalnej karty o 242 końcówkach, którą wkłada się do złącza krawędziowego o nazwie Slot 1 (patrz rysunek dla Pentium 3). Dodatkową zmianą było oddzielenie pamięci podręcznej L2 (ang. *Second Level Cache*) od procesora, a stało się to dlatego, że taki sposób produkcji był znacznie tańszy. Pamięć L2 miała rozmiar 256/512 kB, taktowana była częstotliwością równą połowie taktu zegara procesora i znajdowała się na płytce wraz z procesorem.

Pentium II przetwarzał dane w trzech równoległych dwunastostopniowych potokach, a jądro procesora, jak wspomnieliśmy - jest wykonane w architekturze RISC. W praktyce instrukcje z modelu x86 rozkładane są na proste mikrooperacje i grupowane w centralnym zbiorniku (ang. *Instruction Pool*). Dzięki temu zbiornikowi pobieranie kolejnych kodów x86 z pamięci operacyjnej jest niezależne od ich wykonania. Ze zbiornikiem instrukcji połączony jest układ dyspozytora (ang. *Dispatcher*), który kieruje mikroinstrukcje do właściwych jednostek wykonawczych (ang. *Execute*). Dyspozytor kieruje do wykonania te mikrokody, które aktualnie nie czekają na żadne wyniki pośrednie z innych operacji. Wykonane instrukcje RISC trafiają znowu do zbiornika instrukcji (nie wiadomo, czy układ przewidywania rozgałęzień dobrze określił czy dana instrukcja rzeczywiście jest potrzebna). Instrukcja uznana za wykonaną zostaje "wyrzucona" ze zbiornika i przechodzi na "emeryturę" (ang. *retire*) i dopiero w tym momencie wyniki jej działania są zapisane.

Następny procesor z serii Pentium III (nazwany roboczo Katmai), tak jak jego poprzednik, miał architekturę 32-bitową. Ponadto posiadał zintegrowaną 512 kB pamięć podręczną drugiego poziomu, wykorzystywał poszerzoną 256-bitową szynę BSB (ang. *Back Side Bus*) zapewniającą szybszy transfer wewnętrzny danych. Procesor zasilany był napięciem 1.65/1.7 V (zależnie od modelu). W Pentium III rozszerzono listę rozkazów - dodano ponad 70 rozkazów określanych jako ISSE (ang. *Internet Streaming SIMD Extentions*), czyli wykorzystanie znanych z MMX technik przetwarzania kilku danych z użyciem jednego rozkazu. Co ważne, ISSE potrafi operować także na danych w formacie zmiennoprzecinkowym, co wykorzystywane jest do obliczania obrazów 3D, rozpoznawania mowy, itp...

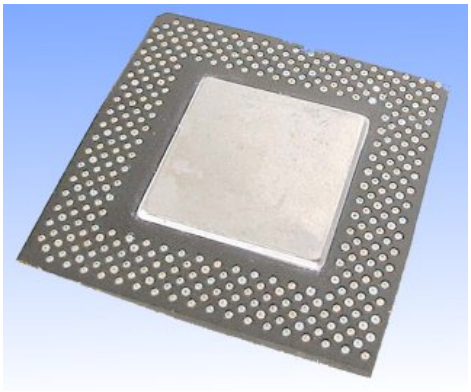


Procesor Pentium III

Równolegle z procesorami Pentium II firma Intel wprowadziła do sprzedaży ich tańszy odpowiednik - serię nazwaną Celeron. Pierwsze procesory nazywano Celeron A Medocino (określenie "medocino" identyfikuje strukturę krzemową). Wyposażone były w zintegrowaną pamięć podręczną L2 o skromnym rozmiarze 128 kB, która była taktowana z pełną prędkością zegara procesora, czyli z prędkością cache L1.

Medocino współpracuje z magistralami 66/100MHz. Taka konstrukcja procesora zapewniła mu dużą wydajność, przy stosunkowo niskim koszcie.

Dodatkowo, Celeron A jako pierwszy dał możliwość "podkręcania" (ang. *overclocking*) częstotliwości pracy magistrali systemowej współpracującej z tym procesorem, a więc częstotliwości pracy procesora (znane są przypadki podkręcania Celeron'a 333 do ok. 700 MHz). Sprawiło to, że Celeron A stał się w swoim czasie rynkowym przebojem i idealną "*maszyną do overclocking'u*".

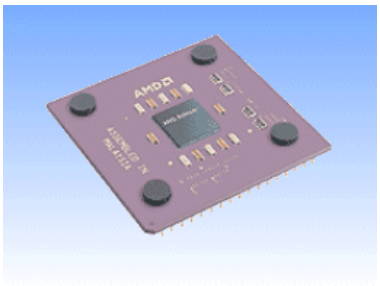


Procesor Celeron

W drugiej połowie lat 90 istniała silna konkurencja na rynku procesorów zgodnych z architekturą x86. Najważniejsi gracze to były firmy VIA (specjalizująca się w wyjątkowo tanich układach) oraz AMD (konkurująca nie tylko ceną ale też wydajnością swoich rozwiązań). AMD między innymi produkował procesory Duron - opracowane głównie z myślą o klientach, którzy potrzebują dużej mocy obliczeniowej za niewielką cenę. Ograniczenie kosztów możliwe stało się dzięki zmniejszeniu ilości pamięci ciche poziomu L2 do 64 KB. Pamięć cache poziomu L1 w procesorze Durom wynosi 128 KB, a więc po raz pierwszy w historii procesor rodziny x86 posiada mniejszy cache poziomu L2 niż poziomu L1. Nie powoduje to jednak znacznego spadku wydajności dzięki zastosowaniu specjalnej organizacji pamięci podręcznej. Wykluczająca się architektura dostępu do pamięci cache (ang. *exclusive cache*) to rozwiązanie, które pozwala procesorowi używać sumarycznej pojemności pamięci cache L1 i L2 tak, jakby to była pamięć poziomu L1 o średnim czasie dostępu 1.5 cyklu zegara.

Zaimplementowana w Duronie interpretacja rozkazów *3DNow!* i *Enchanced 3DNow!*, które wspomagają operacje multimedialne, pozwalała na równoległe przetwarzanie kilku liczb arytmetyki stałoprzecinkowej jak i zmiennoprzecinkowej, a więc zwiększenie szybkości przetwarzania danych - była to realna alternatywa dla wprowadzanych przez Intel równolegle instrukcji SSE.

Innym procesorem AMD był Athlon Thunderbird - jego budowa wewnętrzna była niemal identyczna jak ta, która charakteryzuje procesor AMD Duron. Najbardziej istotną różnicą jest rozmiar pamięci cache L2 działający z pełną szybkością jądra układu, który wynosi 256 KB. Pamięć ta pracuje w trybie *exclusive L2 cache memory*. Procesory Athlon współpracowały z 200/266 MHz magistralą systemową, dzięki czemu możliwe jest dostarczenie do procesora 1.6/2.13 GB danych. Ten wydajny interfejs magistrali zewnętrznej Thunderbirda - EV6 - pochodził od procesorów Alpha firmy DEC.



Procesor Athlon Thunderbird

W 2000 roku firma Intel postanowiła odejść od wywodzącej się od Pentium Pro wewnętrznej architektury RISC. Nowa wewnętrzna architektura o nazwie NetBurst zadebiutowała w procesorach Pentium 4. Teoretycznie mikroarchitektura NetBurst została zaprojektowana pod kątem osiągania bardzo dużych częstotliwości zegara taktującego (między innymi wydłużono potok wykonawczy do 20 etapów, a w kolejnych odsłonach tej architektury – np. w jądrze o kodowej nazwie Prescott – liczył on nawet 31 etapów), w praktyce okazało się, że w seryjnie produkowanych procesorach nigdy nie udało się uzyskać taktowania wyższego niż 4 GHz.

Procesor Pentium 4 był pierwszym od dłuższego czasu, w pełni nowym opracowaniem - a nie modyfikacją poprzednich. Firma Intel - czując na plecach konkurencję AMD, postanowiła kompletnie przeprojektować jądro procesora. Wprowadzony do sprzedaży w końcówce 2000 roku PIV był procesorem 32 bitowym, mogącym pracować z wyjątkowo wysokimi częstotliwościami zegara - w zasadzie limitowanymi jedynie poprzez ograniczenia w możliwości odprowadzenia ciepła.

W styczniu 2004 roku zadebiutowały procesory P4 z zupełnie nowym jądrem "Prescott". Produkowane w technologii 90 nm nowe jądro w pełni uzasadniałoby nazwanie tego procesora P5 - lecz ze względów marketingowych Intel nie zdecydował się na taki krok. Podstawową zmianą było rozszerzenie listy rozkazów procesora o instrukcje 64-bitowe (lista instrukcji x86-64), co spowodowało, że był to pierwszy procesor Intela

przeznaczony na masowy rynek, który możnaby nazwać 64-bitowym. Pozostałe zmiany były mniej rewolucyjne. Poprawiono kolejkowanie rozkazów, co pozwoliło na zwiększenie szybkości przetwarzania, poprawiono wewnętrzną architekturę, co spowodowało dalsze możliwości wzrostu częstotliwości zegara (do 3.8 GHz), oraz wprowadzono kolejną rewolucję - technologię HT (nie we wszystkich modelach).

P4 posiadał nowy mechanizm mający na celu poprawienie wydajności układu. Jest to 8 kB bufor śledzenia wykonywania instrukcji (ang. *Execution Trace Cache*), przechowujący kod x86 w postaci mikrooperacji (nie ma konieczności odkodowywania instrukcji, gdyż są już odkodowane). Ponadto ALU pracuje dwa razy szybciej niż reszta procesora, a więc np. 2.8 GHz dla procesora 1.4 GHz. Mechanizm ten nosi nazwę *Rapid Execution Engine* i umożliwia zakończenie operacji na danych stałoprzecinkowych w połowie cyklu zegarowego.

Najmniej zmienioną częścią P4 Intela była jednostka zmiennoprzecinkowa. Tak jak dla P3 wykorzystywane są dwa kanały do obliczeń zmiennoprzecinkowych. Poszerzony został zestaw instrukcji SSE (nazywane SSE2, oraz dla jąder Presscot SSE3), umożliwiając wykonywanie operacji na liczbach 128 bitowych o podwójnej precyzji oraz na 128 bitowych liczbach stałoprzecinkowych.



Procesor Pentium 4

Dość szybko okazało się, że bez możliwości dalszego przyspieszania zegara mikroarchitektura NetBurst zaczęła wyraźnie przegrywać pod względem wydajności z konkurencyjnymi procesorami Athlon firmy AMD. Mimo to Intel konsekwentnie „obstawał" przy swojej technologii i ją rozwijał. Produkcji procesorów z architekturą NetBurst zaniechano dopiero w 2008 roku. W międzyczasie Intel wrócił do sprawdzonej architektury Pentium III, którą rozwinął i unowocześnił. Poligonem doświadczalnym dla niej były mobilne procesory Intel Core oraz Intel Core Duo, bazujące na jądrze o kodowej nazwie Yonah, na których przetestowano założenia nowej mikroarchitektury Intel Core Microarchitecture – m.in. współużytkowaną pamięć cache oraz strukturę wielordzeniową, wytwarzaną w jednym kawałku krzemu. Układy te stały się podstawą do zaprojektowania procesorów Intel Core 2 Duo (nazwa kodowa Conroe), a później architektury Nehalem oraz kości Intel Core i3, i5 i i7 z najnowszym jądrem Sandy Bridge.

Architektura x86-64

W 2003 roku, wraz z wprowadzeniem na rynek przez firmę AMD układów Opteron (serwery) oraz Athlon 64 na rynku zadebiutowało kolejne rozszerzenie programowej architektury x86 – tym razem do 64-bitów. Co ciekawe, było to pierwsze w historii rozszerzenie modelu programowego, które opracowane zostało przez firmę AMD, a nie jak dotąd przez Intela – choć, gwoli ścisłości, trzeba zaznaczyć, że obie firmy nad nim pracowały wspólnie, jednak to AMD zdecydowała się jako pierwsza je zaimplementować w swoich produktach.

Athlony 64 i serwerowe Opterony wykorzystywały mikroarchitekturę K8 znaną też pod nazwą Hammer – wymienione procesory wykorzystywały jądra odpowiednio o kodowych nazwach Clawhammer i Sledgehammer. Mikroarchitektura Hammer wykorzystywała 64-bitowy model programistyczny o nazwie AMD64, nazywany też x86-64, a w wypadku układów Intela znany jako EM64T, który jest prostym rozwinięciem dotychczas stosowanej w procesorach x86 32-bitowej architektury IA-32.

Rozwinięcie modelu programistycznego x86-64 polega na rozszerzeniu zestawu instrukcji x86 o rozkazy operujące na 64-bitowych operandach. Zwiększona została liczba rejestrów ogólnego przeznaczenia z ośmiu do szesnastu. Dodatkowo liczba 128-bitowych rejestrów SSE używanych w instrukcjach SIMD również zwiększona została z ośmiu do szesnastu. Powiększenie liczby widocznych dla programisty rejestrów pozwoliło na około 5-15% zwiększenie wydajności.

Jedną z przyczyn, która spowodowała pojawienie się potrzeby rozszerzenia możliwości architektury x86 było coraz większe zapotrzebowanie programów na dostępną w systemie pamięć operacyjną. 32-bitowa architektura x86 pozwala na zaadresowanie 4 GB pamięci, z czego dla aplikacji - w zależności od systemu operacyjnego - dostępnych jest najczęściej 2 GB lub 3 GB. Architektura x86-64 teoretycznie mogłaby zdefiniować pełną 64-bitową przestrzeń adresową, czyli 16 EiB (2^{64} bajtów). W praktyce współczesne procesory wykorzystują jednak 48-bitowe adresy wirtualne, co daje 256 TiB przestrzeni adresowej na proces (to i tak 65 536 razy więcej niż 4 GiB w klasycznym 32-bitowym x86). W pierwszych implementacjach AMD64 fizyczna przestrzeń adresowa była rzeczywiście ograniczona do 40 bitów (1 TiB pamięci fizycznej), ale kolejne generacje procesorów AMD i Intela stopniowo zwiększały ten limit do 48 bitów fizycznych (256 TiB), a w serwerowych układach z pięciopoziomowym stronicowaniem – nawet do 52 bitów fizycznych (4 PiB) i 57 bitów wirtualnych (128 PiB). W praktyce ograniczeniem jest dziś raczej liczba gniazd pamięci na płycie głównej i możliwości konkretnego systemu operacyjnego niż sama architektura procesora [Źródło: x86-64].

W architekturze x86-64 przewiduje dwa główne tryby pracy – tryb Long i tryb Legacy. Pierwszy z nich jest podstawowym trybem działania procesora w architekturze x86-64. Jest to kombinacja 64-bitowego trybu pracy i zgodnego wstecz trybu 32-bitowego. W trybie Long 64-bitowy system operacyjny może uruchamiać zarówno programy 64-, jak i aplikacje 32-bitowe oraz programy 16-bitowe – możliwość uruchamiania tych ostatnich zależy jedynie od mechanizmów zawartych w systemie operacyjnym, a nie od samego procesora. Drugi z trybów – tryb Legacy, jest używany przez 16-bitowe i 32-bitowe systemy operacyjne – w tym 32-bitową wersję systemu Windows 7. Jak można się domyślić mogą być w nim używane jedynie programy 16- i 32-bitowe – nie ma w tym trybie możliwości uruchomienia programów 64-bitowych. Procesor zgodny z architekturą x86-64 po włączeniu zasilania jest uruchamiany w trybie Legacy, a mówiąc ściślej w trybie rzeczywistym 8086, i do trybu Long musi zostać przełączony przez uruchomiony program (system operacyjny).

Oczywiście również Intel dysponuje obecnie procesorami 64-bitowymi zgodnymi z architekturą x86-64. Są to układy ze wspomnianą wyżej technologią EM64T (Extended Memory 64 Technology). Projekt wdrożenia przez Intela 64-bitowych rozszerzeń x86-64 znany pod kodową nazwą Yamhill ogłoszony został w lutym 2004 roku na konferencji IDF. Pierwszy procesor Intela z rozszerzeniami EM64T pojawił się w czerwcu 2004 roku – był to serwerowy Xeon z rdzeniem Nocona, który obsługiwał 64-bitowe instrukcje nieoficjalnie. Procesory przeznaczone dla komputerów osobistych z rdzeniem Prescott, produkowane od lutego 2004 roku, miały tę technologię zablokowaną. Oficjalna premiera technologii EM64T nastąpiła w 2005 roku, wraz z premierą procesorów Pentium 4 z jądrem Prescott-2M. Trzecim producentem procesorów zgodnych z programową architekturą x86-64 jest tajwańska firma VIA – procesory począwszy od modelu VIA Nano (nazwa kodowa Isaiah) z 2008 roku.

Wprowadzenie rozszerzeń x86-64 to nie ostatnia ważna zmiana w modelu programowym x86. Od czasu jej wprowadzenia pojawiły się bowiem kolejne rozszerzenia modelu programowego, jak następne wersje SSE (ostatnia z nich to SSE 4.2), rozbudowujące listę instrukcji oraz, przede wszystkim, wektorowe rozszerzenie AVX (Advanced Vector Extensions). W AVX wprowadzono 256-bitowe rejestry, które są dwa razy większe niż wykorzystywane w rozszerzeniach SSE. Nowych rejestrów jest w sumie 16 (osiem dotychczasowych rejestrów SSE wydłużono ze 128 do 256 bitów oraz dodano osiem nowych) i noszą one nazwy YMM0...YMM15. Dodano 19 rozkazów działających wyłącznie na rejestrach YMM oraz czteroargumentowe rozkazy akumulujące wyniki mnożenia wektorów liczb zmiennoprzecinkowych (12 instrukcji), a także sześć rozkazów wspomagających szyfrowanie AES. W sumie zestaw rozkazów w najnowszych procesorach zgodnych z architekturą x86 liczy obecnie 678 instrukcji.

Implementacje tej architektury w procesorach to przede wszystkim w przypadku firmy Intel seria Core 2, ale wcześniej debiutowała w procesorze Pentium IV - w którym zwiększono rozmiar pamięci podręcznej L2 z 512 KB do 1 MB oraz wydłużono potok wykonawczy. W późniejszych wersjach dodano obsługę EDB (seria J) oraz technologię oszczędzania energii (stepping E0).

Nowy rdzeń Prescott wykonany w technologii 0,09 mikrona, 1 MB pamięci podręcznej L2 technologia oszczędzania energii: Enhanced SpeedStep Technology (tylko stepping E0) poprawione bezpieczeństwo dzięki Execute Disable Bit (tylko wersja J). Wprowadzono technologię HyperThreading – (dwa logiczne procesory)zwiększa płynność pracy systemu przy działających równocześnie kilku aplikacjach. Daje to wzrost wydajności o ok. 10-20%

Natomiast jeśli chodzi o sam procesor, to pierwsza wersja Prescotta z technicznego punktu widzenia okazała się klapą. Przejście na technologię 90 nm spowodowało liczne problemy, z którymi Intel nie zdołał się na czas uporać. W efekcie Prescott bardzo się grzał, nie będąc jednocześnie specjalnie szybszym od swojego poprzednika opartego na jądrze Northwood - nawet pomimo dwukrotnego zwiększenia pamięci cache.

Na płytach z Chipsetami do tego procesora po raz pierwszy zastosowano nową magistralę PCI-Express. Z myślą o zwiększonym poborze energii wprowadzono nowe, beznóżkowe gniazda LGA775.

Nowszą serią były procesory rodziny Core / Centrino. Były to pierwsze produkty wielordzeniowe przeznaczone do zastosowania w urządzeniach przenośnych. Z punktu widzenia użytkownika dwa rdzenie w notebooku oznaczają "tylko" dalszy, choć istotny wzrost szybkości laptopów. Dla inżynierów projekt stanowił nie lada wyzwanie. To, co od biedy ujdzie w stacjonarnym pececie, w urządzeniu mobilnym może być wadą nie do zaakceptowania: kilka watów poboru mocy więcej w "zwykłym" procesorze da się zrekompensować odpowiednim chłodzeniem; w notebookowym układzie taka różnica przełoży się na fiasko całego przedsięwzięcia. Intelowi udało się pogodzić ogień z wodą, i seria Core 2/Centrino stała się rynkowym sukcesem.

Oprócz zmiany nazwy CPU z Pentium na Intel Core wprowadzona została nowa numeracja modeli układu. Zamiast trzycyfrowego kodu procesor oznaczany jest literą symbolizującą klasę poboru mocy i czterocyfrową liczbą będącą wskaźnikiem wydajności.

Następcą Core był Core 2 Duo. Mimo zbliżonej nazwy do Core Duo - to jest zupełnie nowy procesor, przeznaczony nie tylko do Laptopów, lecz także do systemów stacjonarnych.



W tych procesorach zastosowano dość krótki potok wykonawczy, który składa się z 14, a nie 31 etapów, jak w Pentium 4. Obniżyło to maksymalną częstotliwość pracy i ograniczyło zużycie energii, a procesor się nie przegrzewa. Ponieważ jego wydajność zależy od liczby instrukcji wykonywanych w każdym takcie zegara sterującego i od częstotliwości, bardzo skuteczne okazało się zastosowanie czwartej jednostki wykonawczej.

Aby została wykorzystana, należało zastosować układ niekolejnych pobrań z pamięci (Memory Disambiguation), dzięki czemu dane do przetwarzania są w pamięci podręcznej i na czas trafią do potoku wykonawczego. Poważną modyfikacją jest wykonywanie wszystkich instrukcji SSE i SSE2 w jednym takcie zegara, zamiast w dwóch. Wobec braku zintegrowanego kontrolera pamięci bardzo istotną funkcję pełni współdzielona przez rdzenie pamięć podręczna drugiego poziomu. W zależności od wykonywanej aplikacji może być w całości przydzielona jednemu rdzeniowi lub w dowolnych proporcjach obu. Pamięć ta połączona jest z rdzeniami 256-bitową szyną danych, co pozwala jednocześnie przesłać 4 bajty danych.

W tańszych procesorach Core 2 Duo E6400 (2,13 GHz) i E6300 (1,86 GHz) pamięć cache L2 była ograniczona do 2 MB. W szybszych wersjach procesorów: Core 2 Due Extreme X6800 (2,93 GHz), Core 2 Duo E6700 (2,66 GHz) oraz Core 2 duo E6600 (2,4 GHz) miała wielkość 4 MB. Pamięć podręczna o wielkości 4 MB składa się z około 270 milionów tranzystorów co stanowi ponad 90 procent łącznej liczby tranzystorów w układzie. Jednak dla kosztów wytwarzania układów istotniejsze jest to, że cache zajmuje jedynie około połowy powierzchni struktury krzemowej, która przy produkcji w procesie technologicznym 65 nm zajmuje jedynie 143 mm2.

W uzyskaniu dużej wydajności przez nowe procesory pomagają także inne usprawnienia. Pamięć podręczna L1 procesorów przechowuje instrukcje w postaci wstępnie zdekodowanej. Bufory dla pobieranych danych są większe, więc mogą przechowywać więcej informacji. Instrukcje x86 rozkładane są na tzw. makro operacje, dzięki czemu mogą być łączone w celu lepszego wykorzystania przepustowości połączeń. Przykładem takiego połączenia jest para Cmp+Jmp (porównaj i skocz), która bardzo często występuje w kodzie programów.

Bardzo istotną modyfikacją jest wprowadzenie układu śledzącego wskaźnik stosu (Extender Stack Pointer). Wszelkie obliczenia na tym rejestrze adresowym nie zajmują jednostek obliczeniowych procesora, a jednocześnie odpowiedni adres jest dostępny zawsze wtedy, gdy jest potrzebny bez konieczności oczekiwania na wyliczenie go.

AMD Athlon 64 to pierwsze procesory 64-bitowe (AMD64) przeznaczone dla wydajnych stacjonarnych zestawów komputerowych. Procesory te zależnie od modelu, mają od 512 do 1MB wbudowanej pamięci podręcznej L2 oraz na gnieździe Socket 754 i Socket 939. Taktowanie procesora było od 1800(2800+) do 2400MHz(4000+) lub 2600MHz(4000+) dla AM2.

Zwiększona przestrzeń adresowa pojedynczego procesu umożliwia procesowi odwoływanie do 256 TB pamięci wirtualnej. Procesory AMD64 mogą zaadresować do 1TB fizycznej pamięci operacyjnej. W celu maksymalnego skrócenia dostępu do pamięci operacyjnej procesory tej firmy mają zintegrowany kontroler pamięci. Układ oznaczony symbolem Athlon 64 i Sempron 3300+ na gnieździe Socket 754. Podobnie jak procesory Intela obsługuje instrukcje multimedialne SSE3. Procesor jest wykonany w technologii 90nm pracujący z częstotliwością 2.0 GHz i pamięcią podręczną L2 128Kb. Procesor AMD ma specjalną szynę do komunikacji z pamięcią RAM a z resztą urządzeń takich jak kontrolery PCI-E , podłączony jest przez magistralę HyperTransport pracującą z częstotliwością do 2.6 GHz(dla wersji 3.0). Dla przykładu HyperTransport 2.0 pracuje przy 1.4 GHz i ma przepływność 22.4GB/s.

Pełne wykorzystanie mocy obliczeniowej tych procesorów jest możliwe dopiero po zainstalowaniu 64 bitowego systemu operacyjnego

Opracowana była również wersja FX procesorów AMD - 64-bitowych dla stacji roboczych, przeznaczona dla gier lub bardziej skomplikowanych obliczeń graficznych. Procesory FX były porównywalne pod względem wydajności z Intel Core 2 Duo, przy zazwyczaj niższej cenie.

Firma AMD także wprowadziła na rynek procesory wielordzeniowe (tak jak Core 2) - np. dwurdzeniowy procesor AMD Athlon 64 X2, składał się z dwóch jednostek obliczeniowych połączonych w jednym układzie, wykonujących obliczenia na dwóch strumieniach danych.

Równolegle rozwijana była seria AMD Operton - produktów firmy AMD przeznaczony dla stacji roboczych oraz serwerów. Zastosowana w nich technologia AMD64 z Architekturą Direct Connect zwiększyła całkowitą wydajność systemu, eliminując tradycyjne wąskie gardła charakterystyczne dla dotychczasowych systemów. Dotychczas magistrala FSB ograniczała i przerywała przepływ danych. Przerwy w przepływie danych oznaczają mniejszą skalowalność systemu. W architekturze Direct Connect nie ma magistrali FSB. Zamiast tego rdzenie procesora, kontroler pamięci oraz układ We-Wy są połączone bezpośrednio ze sobą i komunikują się z częstotliwością zegara procesora.

Współczesne procesory do komputerów osobistych i przenośnych

Jak już wspomnieliśmy, współcześnie na rynku liczą się dwie firmy: Intel i AMD. W przypadku firmy Intel podstawową rodziną procesorów dla typowych zastosowań jest Core - występująca w trzech seriach (Core i3, i5 oraz i7; w nowszych generacjach także Core i9). Są to wielordzeniowe (od 2 do 4 rdzeni i więcej) procesory 64 bitowe, zazwyczaj (choć nie zawsze) mające wbudowany prosty układ graficzny. Pełne oznaczenie procesora zawiera serię, 3–4 cyfry oraz sufiks literowy. Najczęściej pierwsza cyfra (lub dwie) oznacza generację procesora, pozostałe cyfry to numer modelu/SKU, natomiast szczegóły dotyczące przeznaczenia i zużycia energii określają przede wszystkim litery na końcu nazwy (np. U – niskie zużycie energii, H – wysokowydajna grafika mobilna, K – odblokowany mnożnik, F – brak zintegrowanej grafiki itd.) [Źródło: Intel Processor Numbers]. Warto też wiedzieć, że w połowie lat 20. XXI wieku Intel rozpoczął rebranding nazw, przechodząc stopniowo z oznaczeń „Core i5/i7/i9” na „Core 5/7/9” oraz „Core Ultra” – dlatego w opisach nowszych modeli można spotkać oba schematy nazewnictwa równolegle [Źródło: Intel Core Ultra].

W momencie aktualizacji tego podręcznika typowy komputer stacjonarny z procesorem Intela opiera się zwykle na gnieździe LGA1700 oraz procesorach 12., 13. lub 14. generacji Core (Alder Lake, Raptor Lake i ich odświeżenia), łączących rdzenie „wydajne” (P-cores) i „energooszczędne” (E-cores) w jednej strukturze. W segmencie mobilnym coraz częściej spotyka się procesory Core Ultra z wbudowanymi jednostkami NPU (Neural Processing Unit) do przyspieszania obliczeń związanych ze sztuczną inteligencją [Źródło: LGA1700, Intel Meteor Lake]. Po stronie AMD rolę głównej platformy dla komputerów stacjonarnych przejęło gniazdo AM5 z procesorami Ryzen 7000/8000, obsługujące pamięci DDR5 i interfejs PCIe 5.0, przy jednoczesnym utrzymywaniu zgodności wstecznej i nowych modeli dla starszego gniazda AM4 (np. Ryzen 5 5600F, 5500X3D) [Źródło: Socket AM5, AMD Ryzen].

W założeniach, seria i3 miała być serią najmniej wydajną, i5 środkową, natomiast i7 najbardziej wydajną. Lecz ze względu na wewnętrzne zróżnicowanie każdej serii przyjęcie założenia, że np i7 jest zawsze szybszy niż i5 jest błędne - są procesory i5 o zdecydowanie wyższej wydajności niż najsłabsze modele i7, przy porównywaniu należy uwzględniać wszystkie parametry procesora, a nie tylko jego serię.

Oprócz produktów z rodziny Core, Intel oferuje również rozwiązania serwerowe, czyli rodzinę Xeon, składającą się z czterech serii (od najbardziej wydajnej): E7, E5, E3, Phi - oznaczanej wg zasad zbliżonych, lecz nie identycznych, do rodziny Core. Inną rodziną procesorów firmy Intel jest Atom - budżetowe rozwiązanie o bardzo niskim zużyciu energii, przeznaczone do netbooków, tabletów czy smartfonów. Są to wyraźnie mniej wydajne procesory, często ciągle 32 bitowe, ale za to nie wymagające aktywnego chłodzenia.

Firma AMD oferuje zarówno procesory klasyczne (seria FX do komputerów ogólnego przeznaczenia oraz Operton do rozwiązań serwerowych) oraz zintegrowane z dość silnymi kartami graficznymi jednostki nazywane APU. Wydaje się, iż od jakiegoś czasu AMD pogodziło się z rolą "tego drugiego" producenta układów w architekturze x86-64, w związku z tym w dziedzinie klasycznych procesorów konkuruje głównie ceną, nie mając ambicji zaprezentowania najszybszych procesorów na rynku. W zamian za to stawia na nowe pomysły - tym jest właśnie APU i wewnętrzne elementy jego architektury. W najnowszej generacji, zwanej kodowo Kaveri, takim innowacyjnym elementem jest obsługa HSA. HSA to sposób konstruowania sprzętu komputerowego, w którym procesory każdego rodzaju wchodzące w skład jednego chip-a są równorzędne: mają równy dostęp do wspólnej pamięci operacyjnej oraz mogą sobie nawzajem przekazywać zadania. Na razie mamy na myśli rdzenie procesora oraz układ graficzny; w przyszłości będą mogły do nich dołączyć inne typy maszyn obliczeniowych. To znaczące odejście od tradycyjnych systemów, w których centralny procesor (CPU) jest „szefem” układu, procesor graficzny ma własny obszar pamięci, a komunikacja między nimi wiąże się z tłumaczeniem adresów i przekazywaniem obszarów pamięci pod kontrolę to jednego, to drugiego procesora. Tutaj mamy zaimplementowane rozszerzenie architektury NUMA do standardu zwanego hUMA - czyli pamięć jest bezpośrednio wzpółdzielona pomiędzy wszystkie jednostki wykonawcze każdego typu implementowane w procesorze. Ponadto - w odróżnieniu od grafiki wbudowywanej w procesor przez firmę Intel, w przypadku AMD APU zintegrowane układy graficzne są zdecydowanie bardziej wydajne, pozwalające w wielu wypadkach nawet na uruchamianie współczesnych gier.

Ze względu na bardzo szybki postęp w tej dziedzinie, zamieszczanie w podręczniku opisów konkretnych modeli współczesnych procesorów jest z góry skazane na szybkie przedawnienie. Uważamy, że zdecydowanie lepiej będzie polecić Wam kilka źródeł w internecie, gdzie zawsze znajdziecie najbardziej aktualne informacje:

- www.benchmark.pl - strona poświęcona głównie testom sprzętu komputerowego, zawierająca także ciekawe opisy natury ogólnej
- www.chip.pl - strona czasopisma komputerowego, w której zawsze możecie znaleźć aktualne rankingi podzespołów PC
- www.notebookcheck.pl - kolejna strona poświęcona testom sprzętu komputerowego, w tym wypadku głównie przenośnego - także zawierająca często aktualizowane zestawienie procesorów
- www.pclab.pl - kolejna strona poświęcona testom sprzętu komputerowego

Architektura ARM

Architektura ARM (Advanced RISC Machine, pierwotnie Acorn RISC Machine) – 32-bitowa oraz 64-bitowa architektura i model programowy procesorów, będąca od początku architekturą typu RISC. Mimo iż nie widzicie tego na co dzień, procesory z tą architekturą są jednymi z najczęściej stosowanych procesorów na świecie. Używa się ich w dziesiątkach tysięcy rozwiązań wbudowanych (dyskach twardych, telefonach komórkowych, routerach, kalkulatorach, zabawkach dziecięcych, itp ....). Obecnie procesory oparte na architekturze ARM dominują w urządzeniach mobilnych (smartfony, tablety, wearables) oraz w ogromnej części systemów wbudowanych i IoT – standaryzują znaczną większość rynku 32- i 64-bitowych mikrokontrolerów i mikroprocesorów w tych zastosowaniach [Źródło: ARM architecture]. Wraz z przejściem na 64-bitowe warianty ARMv8/ARMv9 w smartfonach i tabletach rośnie też znaczenie ARM w segmencie serwerów (np. AWS Graviton, Ampere Altra) oraz komputerów osobistych (laptopy Apple z układami M-serii) [Źródło: AWS Graviton, Apple silicon]. 32-bitowe rdzenie ARM pozostają ważne głównie w bardzo tanich, prostych urządzeniach i aplikacjach o niskich wymaganiach pamięciowych.

Od opisywanej wcześniej architektury x86 różni się bardzo wieloma względami technicznymi oraz fundamentalnie innym podejściem biznesowym. Stojąca za projektem ARM firma ARM sama nie produkuje własnych procesorów. Zamiast tego tworzy ich projekty i sprzedaje je innym firmom wg dwóch schematów:

- gotowe do implementacji projekty elektroniki (jądro Cortex) do osadzenia w swoich układach. Z tej pierwszej możliwości korzystają takie marki, jak Samsung (seria układów Exynos), Nvidia (seria układów Tegra), Broadcom (seria układów BCM, które są montowane w Nokiach z Symbianem), ST-Ericsson (seria NovaThor, często spotykana w smartfonach ze średniego segmentu) i Texas Instruments (seria OMAP).
- licencję na tworzenie układów podobnych do swoich i kompatybilnych z wybranym zestawem instrukcji. Z tej drogi, znacznie trudniejszej, w tym momencie korzysta jedynie Qualcomm (sprzedający układy z rdzeniami Snapdragon i Krait) i Apple (architektura Swift, która zadebiutowała wraz z iPhone'em 5), choć wiadomo też, że Nvidia od jakiegoś czasu konstruuje coś własnego, co może trafić do kolejnych generacji Tegr.

Projektowanie pierwszego procesora ARM rozpoczęło się w 1983 roku, jako projekt rozwojowy brytyjskiej firmy Acorn Computers Ltd. Grupa inżynierów kierowana przez Rogera Wilsona i Steve'a Furbera rozpoczęła projektowanie jądra będącego ulepszoną wersją procesora MOS 6502 firmy MOS Technology. Pierwsza wersja testowa, nazywana ARM1, opracowana została w 1985 roku, a rok później ukończono wersję produkcyjną ARM2. ARM2 wyposażony był w 32-bitową szynę danych, 26-bitową przestrzeń adresową oraz w szesnaście 32-bitowych rejestrów. Był to w tym czasie najprostszy szeroko stosowany 32-bitowy mikroprocessor, zawierający tylko 30 tysięcy tranzystorów. Prostota wynikała głównie z braku mikrokodu i, jak w większości procesorów w tym czasie, braku cache. ARM2 miał z tego powodu bardzo niski pobór mocy i jednocześnie szybkość przetwarzania większą od procesora Intel 80286. Następna wersja ARM3 produkowana była z 4KB cache, co jeszcze bardziej poprawiło wydajność.

W późnych latach osiemdziesiątych firma Apple Computer rozpoczęła współpracę z Acorn Computers w projektowaniu nowszej wersji jądra ARM. Projekt był na tyle istotny, że Acorn wydzielił grupę projektową tworząc w 1990 roku nową firmę o nazwie Advanced RISC Machines (ARM Ltd.). Wynikiem tej współpracy był procesor ARM6, udostępniony w roku 1990. Firma Apple użyła opartego na ARM6 procesora ARM610 w palmtopie (PDA) o nazwie Apple Newton. Jądro procesora ARM6 zawiera około 35 tysięcy tranzystorów i jest tylko niewiele większe od jądra ARM2 (30 tysięcy tranzystorów). Dzięki swej prostocie jądro ARM może być łączone z dodatkowymi blokami funkcjonalnymi, tworząc w jednej obudowie, mikroprocesor dostosowany do konkretnych wymagań. Jest to możliwe, gdyż podstawą działalności ARM Ltd. jest sprzedaż licencji na zaprojektowane jądra. Dzięki temu powstały także mikrokontrolery oparte na architekturze ARM.

Firma DEC zakupiła licencję na architekturę ARM i na jej podstawie zaprojektowała procesor StrongARM. Przy częstotliwości 233 MHz procesor ten pobierał tylko 1W mocy (najnowsze wersje StrongARM pobierają znacznie mniej). Projekt ten został następnie przejęty przez firmę Intel, na podstawie umowy procesowej między obiema firmami. Dla Intela była to szansa na zastąpienie przestarzałej architektury i960 nową architekturą StrongARM. Na podstawie StrongARM Intel zaprojektował bardzo wydajny mikroprocesor o nazwie XScale.

Zgodnie z założeniami architektury RISC, rozkazy procesorów ARM są tak skonstruowane, aby wykonywały jedną określoną operacje i były przetwarzane w jednym cyklu maszynowym. Interesującą zmianą w stosunku do innych architektur jest użycie 4-bitowego kodu warunkowego na początku każdej instrukcji. Dzięki temu każda instrukcja może być wykonana warunkowo. Ogranicza to przestrzeń dostępną, na przykład, dla instrukcji przeniesień w pamięci, ale z drugiej strony nie ma potrzeby stosowania instrukcji rozgałęzień dla kodu zawierającego wiele prostych instrukcji warunkowych. Inną unikatową cechą zestawu instrukcji procesora ARM jest łączenie operacji przesunięcia i obrotu bitów w rejestrze z instrukcjami arytmetycznymi, logicznymi, czy też przesłania danych z rejestru do rejestru. Przedstawione cechy powodują, że typowy program zawiera mniej linii kodu niż w przypadku innych procesorów RISC. W rezultacie jest mniejsza liczba operacji pobrania/zapisania argumentów instrukcji, więc potokowość jest bardziej efektywna. Pomimo że procesory ARM są taktowane zegarem o stosunkowo niskiej częstotliwości są konkurencyjne w stosunku do znacznie bardziej złożonych procesorów.

Zestaw instrukcji i architektury ARM doczekały się już ośmiu generacji. W tym momencie na rynku dominują procesory kompatybilne z architekturą ARMv7 lub jej rozszerzeniem, nazywanym czasem ARMv7s, ARMv8 jest wdrażane do produkcji i na rynku powinno się pojawić już niedługo. ARM, tworząc nową rodzinę procesorów i związany z nią zestaw instrukcji, określa tylko, co będą umiały, ale nie wskazuje dokładnie, w jaki sposób ma to zostać osiągnięte. Dlatego w ramach jednej rodziny ARM projektuje kilka różnych rdzeni, różniących się wydajnością, możliwościami i zastosowaniem. Pierwszym przedstawicielem układów ARMv7 był rdzeń Cortex-A8, kojarzony głównie z pierwszą generacją Samsungów Galaxy S (miały one chip o nazwie Hummingbird, który składał się z jednego rdzenia Cortex-A8 taktowanego z częstotliwością 1 GHz i układu graficznego PowerVR SGX540) i z tym, że przyniósł spory wzrost wydajności w porównaniu z poprzednikiem. Po nim powstał Cortex-A9, spotykany teraz w większości smartfonów w konfiguracjach dwu- lub czterordzeniowych. Główną różnicą w stosunku do Cortex-A8 było to, że A9 umożliwiała tworzenie układów wielordzeniowych oraz nauczyła się wykonywać niektóre instrukcje poza kolejnością (ang. Out-of-Order Execution), co dało kilkunastoprocentowy wzrost wydajności. ARM zaprojektowało też rdzenie Cortex-A5 (spotykane na przykład w HTC Desire X) i Cortex-A7, które są kompatybilne z instrukcjami ARMv7s, ale mniej wydajne (na przykład nie umieją wykonywać instrukcji poza kolejnością) i przystosowane do tańszych i mniejszych układów scalonych.

Najwydajniejszym i najnowszym rdzeniem należącym do rodziny ARMv7 (z rozszerzeniami ARMv7s) jest Cortex-A15 Choć nadal należy on do siódmej generacji procesorów ARM, to poczyniono w nim bardzo dużo zmian w porównaniu z Cortex-A9. Cortex-A15 to rdzeń trójpotokowy, w którym całkowicie zmieniono i poszerzono część zajmującą się dekodowaniem i kolejkowaniem instrukcji, zwiększono liczbę jednostek wykonawczych, rozbudowano część procesora odpowiedzialną za dostarczanie danych i komunikację z pamięcią, itp.

A gdzie w tym wszystkim znajdują się procesory Apple i Qualcomma? Architektury Apple Swift i Qualcomm Krait to coś pomiędzy Cortex-A9 a Cortex-A15. Również są one trójpotokowe, tak jak Cortex-A15 (i w przeciwieństwie do dwupotokowego Corteksa-A9), ale nie mają aż tak rozbudowanych jednostek wykonawczych i układów wejścia/wyjścia, przez co mają mniejszą wydajność, ale też mniejsze zapotrzebowanie na energię, i są zbudowane z mniejszej liczby tranzystorów. Nowsze generacje rdzeni tych firm (Apple A-serii, M-serii oraz kolejne linie Snapdragonów Prime/Gold/Silver) kontynuują tę ideę – są to własne, rozwijane przez producentów implementacje architektury ARM, często znacznie wyprzedzające referencyjne rdzenie Cortex pod względem wydajności na wat [Źródło: Apple silicon, Qualcomm Snapdragon].

Wewnętrznie wszystkie te jednostki są zdecydowanie mniej skomplikowane niż współczesni reprezentanci architektury x86, i też mniej wydajne. Jednak stosunek stopnia skomplikowania (i co za tym idzie - ceny) do wydajności zdecydowanie przemawia za ARM - co dostrzegają też producenci tacy jak AMD, opracowując wersje procesorów ARM dla serwerów (np. AMD Operton A1100).

Zamiast podsumowania

Jak widzicie - świat procesorów jest całkiem skomplikowany. Co więcej - rozwiązania techniczne w nich zmieniają się szybciej niż w innych dziedzinach techniki. Rewolucja porównywalna z wprowadzeniem układów Common Rail w samochodach, w dziedzinie konstrukcji procesorów trafia się raz na dwa / trzy lata. Z tego względu nie bardzo wiadomo, jak będzie rozwijał się świat procesorów. Z jednej strony - mamy procesy utrzymujące przy życiu skomplikowaną architekturę x86 w postaci istniejącego oprogramowania, z drugiej - rosnącą rolę zastosowań

kieszonkowych, przenośnych i innych. Dobrym przykładem tych procesów jest przejście firmy Apple w komputerach Mac na własne procesory ARM (rodzina Apple M), co pokazało, że także w segmencie wydajnych laptopów i stacji roboczych alternatywne architektury mogą skutecznie konkurować z x86 [Źródło: Apple silicon]. Z drugiej strony Intel i AMD intensywnie rozwijają swoje układy x86-64 (w tym specjalizowane jednostki NPU do przetwarzania zadań AI), więc przyszłość będzie prawdopodobnie należała do współistnienia kilku różnych architektur w różnych klasach urządzeń. Jedyne co możemy wam doradzić - to na bieżąco uzupełniać wiedzę o procesorach i kierunkach ich rozwoju korzystając z sieci internet.

1.2. Pamięć operacyjna

Elektroniczne pamięci półprzewodnikowe, które są stosowane w systemach komputerowych, zazwyczaj należą do jednej z dwóch grup:

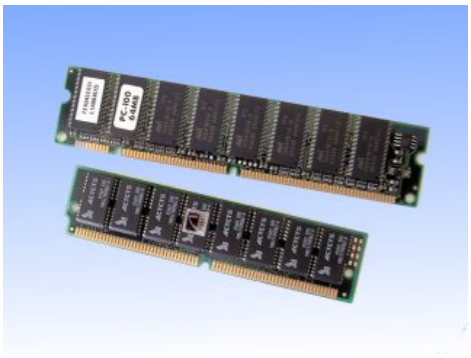
- DRAM (ang. Dynamic Random Access Memory) - ten rodzaj pamięci o dostępie swobodnym przechowuje każdy bit danych w oddzielnym kondensatorze. Ze względu na to, że nośnikiem informacji są kondensatory (w uproszczeniu komórka składa się z kondensatora i tranzystora sterującego, tzw. 1T1C), które samoistnie rozładowują się, zawartość pamięci musi być okresowo odświeżana. Odświeżanie polega na cyklicznym odczytaniu i ponownym zapisaniu tej samej informacji (realizowanym automatycznie przez układ kontrolera pamięci), aby skompensować samorozładowanie kondensatorów. Ponadto typowy odczyt z pamięci DRAM ma charakter „niszczący" w tym sensie, że podczas odczytu ładunek kondensatora jest zaburzany, więc odczytana wartość musi zostać ponownie zapisana, by nie została utracona. Zaletą tego typu pamięci jest natomiast niski koszt i możliwość uzyskania wysokiego stopnia upakowania (dużej ilości bitów pamięci na jednostkę powierzchni). Pamięci DRAM są podstawowym typem stosowanym jako pamięć operacyjna.
- SRAM (ang. Static Random Access Memory), przechowuje dane tak długo, jak długo włączone jest zasilanie, w odróżnieniu od pamięci typu DRAM, która wymaga okresowego odświeżania. Każdy bit przechowywany jest w pamięci SRAM w układzie zbudowanym z tranzystorów tworzących przerzutnik oraz z tranzystorów sterujących (klasyczna komórka SRAM często ma 6 tranzystorów). Taka struktura umożliwia znacznie szybsze odczytanie bitu niż w pamięci typu DRAM, oraz w przeciwieństwie do pamięci DRAM nie wymaga odświeżania. Pamięci SRAM wykorzystywane są w szybkich pamięciach podręcznych cache wbudowanych w procesor lub instalowanych na płytach, ponieważ nie wymagają one dużych pojemności (SRAM jest wyraźnie mniej „gęsta" od DRAM), ale w zamian oferują znacznie krótsze czasy dostępu. Dokładne proporcje (np. „ile razy") zależą od technologii i konkretnej implementacji, dlatego najlepiej traktować to jako porównanie jakościowe: SRAM szybka, DRAM pojemna i tania

Ponieważ zazwyczaj pamięć cache jest wbudowana w procesor, natomiast pamięć operacyjna jest wybierana spośród dostępnych na rynku modułów - to w dalszej części lekcji skupimy się na typach pamięci DRAM.

FPM RAM

FPM RAM (ang. *Fast Page Mode RAM*) jest najstarszym rodzajem pamięci. Jest to pamięć typu asynchronicznego, w której sygnały sterujące matrycą komórek pamięci (sygnały te to: *RAS: Row Adress Strobe* - wybór wiersza matrycy pamięci oraz *CAS: Column Adress Strobe* - wybór kolumny matrycy pamięci) generowane są niezależnie od taktów zegara. Tak więc, informacja na wyjściu ukazuje się po czasie wynikającym z konstrukcji układu. Dodatkowo, występują problemy ze zsynchronizowaniem się taktów zegara systemowego i taktów pamięci

Termin "Fast" odnosi się do faktu, iż pamięci te umożliwiają szybszy dostęp do danych znajdujących się na jednej stronie pamięci. Niegdyś pamięci te były montowane "na pokładzie" 386-óstek i 486-óstek w postaci 72-pinowych modułów SIMM (ang. *Single In-line Memory Module*).



Porównanie pamięci typu SIMM i DIMM

SDRAM

Pamięć SDRAM (ang. *Synchronous DRAM*), podobnie jak pamięć typu FPM, jest pamięcią typu DRAM. Pamięć ta pracuje z częstotliwością zewnętrznej magistrali systemowej (a więc synchronicznie) i charakteryzuje się czasem dostępu rzędu 10 ns. SDRAM-y wyróżnia ponadto wysoka teoretyczna przepustowość danych - 800 MB/s dla kości typu PC-100 i 1064 MB/s dla PC-133. Pamięci SDRAM są wykonywane w postaci 168-pinowych modułów DIMM (ang. *Dual In-line Memory Module*), obecnie zasilanych napięciem 3,3 V.

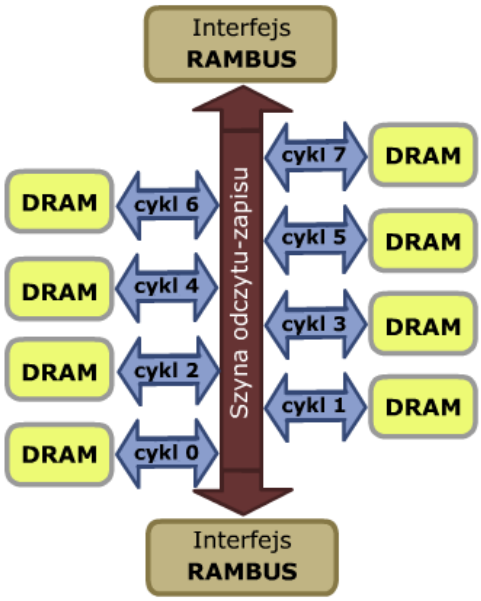
DDR SDRAM

Pamięć DDR SDRAM (ang. *Double Data Rate DRAM*) jest ulepszoną wersją swojej poprzedniczki. Ulepszenie polega na możliwości przesyłania danych na obydwu zboczach sygnału zegarowego. Charakteryzują się bardzo dużą przepustowością - 2.1 GB/s dla DDR SDRAM pracujących efektywnie z częstotliwością 266 MHz.

Podwojenie częstotliwości pracy nie jest jedyną nowinką związaną z pamięciami DDR. Pamięci tego typu posiadają usprawniony mechanizm synchronizacji oraz buforowania danych. Pamięć DDR wykonana jest w postaci 184-pinowych modułów DIMM.

RDRAM

Pamięci RDRAM (ang. *Rambus Direct RAM*) są układami podobnymi do pamięci DRAM. W układach tych matryca pamięci jest podzielona na 8 niezależnych bloków. Każda z części jest odczytywana z pewnym opóźnieniem, wynikającym z częstotliwości zegara. W jednym takcie zegara jest więc odczytana informacja z pojedynczego bloku. Przy kolejnych cyklach pobierane są kolejne dane i dopiero po odczytaniu wszystkich bloków, dane wysyłane są na zewnątrz w postaci pojedynczego pakietu.



Pobieranie danych z poszczególnych banków przez pamięci Rambus

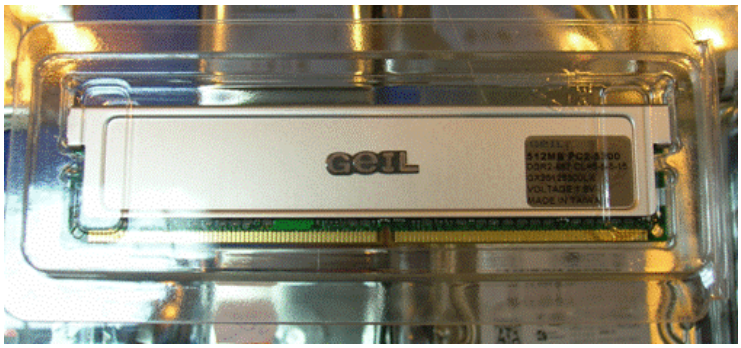
Ważną zaletą pamięci RDRAM jest fakt, że umożliwiają one duże transfery - 1,6 GB/s dla pamięci współpracującej z 16-bitową szyną danych z częstotliwością 400 MHz (efektywnie 800 MHz - informacje przesyłane są na obu zboczach sygnału zegarowego). Podwojenie kanału Rambus (do którego wsparcie oferuje chipset i840) daje przepustowość 3,2 GB/s. Olbrzymia przepustowość pamięci Rambus spowodowała wydłużenie czasu dostępu do danych.

Z wytwarzaniem pamięci RDRAM wiąże się wiele problemów technologicznych, jak np. eliminacja efektu linii długiej, zakłócającego poprawność przesyłania informacji. Wymusza to rezygnację z możliwości zainstalowania więcej niż dwóch modułów pamięci RDRAM na płycie głównej. Pamięć RDRAM wytwarzana jest w postaci modułów RIMM (ang. *Rambus In-line Memory Module*).

Rozwiązania historyczne

W tym miejscu zbierzemy w jedną grupę standardy, które były szeroko wykorzystywane w komputerach osobistych w poprzednich generacjach sprzętu, ale dziś spotyka się je głównie w komputerach starszych lub specjalistycznych. Dla wielu z nich można jeszcze znaleźć moduły na rynku wtórnym, jednak przy modernizacji współczesnych PC zwykle rozważamy już nowsze standardy.

DDR2 (Double Data Rate 2) to następca DDR, w którym zachowano tę samą ogólną zasadę „podwójnego” przesyłu danych na obu zboczach sygnału zegarowego, a jednocześnie zwiększono możliwości uzyskania wyższych efektywnych szybkości transmisji. Historycznie spotykało się moduły m.in. DDR2 400 (PC 3200), DDR2 533 (PC4200), DDR2 667 (PC5300) czy DDR2 800 (PC6400).



Moduł pamięci DDR2 (przykład)

Pamięci te przy odpowiednim parowaniu (dobraniu dwóch takich samych modułów) mogą pracować w trybie Dual-Channel. Jest to zwiększenie szerokości z 64-bitowej na 128-bitową szyny dostępu do pamięci RAM. Obsługę taką zapewniają odpowiednie chipsety płyt głównych. Uzyskuje się teoretycznie zdwojoną przepustowość MB/s przy tej samej częstotliwości taktowania pamięci w stosunku do pojedynczego modułu o tej samej łącznej pojemności.

DDR3 RAM

DDR3 jest kolejnym rozszerzeniem standardu DDR RAM. Standardowe napięcie zasilania dla DDR3 wynosi 1,5 V, a istnieją też odmiany energooszczędne, np. DDR3L (1,35 V) - będące wersją niskonapięciową DDR3.
Ogólnie pamięć DDR3 jest bardziej wydajna niż DDR2, przy czym ze względów marketingowych na sprzedawanych w sklepach modułach podaje się szybkość sygnału DDR a nie częstotliwość zegara. Tak więc oferowano pamięci DDR - 266, 333 i 400 MHz, DDR2 - 533, 667, 800 i 1066 MHz a DDR3 - 1333 MHz i szybsze. Do oznaczenie standardu używa się natomiast jeszcze bardziej imponujących wartości podających przepustowość pamięci. Wyliczamy je według wzoru - (64 bity * 2 * częstotliwość pracy (MHz))/8. Tym samym pamięci DDR2 800 MHz to PC2-6400 (przepustowość 6400 MB/s), zaś DDR3 1600 MHz to PC3-12800.
Błędem byłoby przyjęcie, że teoretyczne uzyskiwane przez moduły RAM przepustowości są jednoznaczne z ich "wydajnością". Dzieje się tak za sprawą czasów dostępu (CL - CAS Latency) czyli czasu jaki mija między wysłaniem przez kontroler pamięci żądania dostępu do jej określonej kolumny a odczytaniem danych z tej kolumny. Im krótszy czas dostępu (niższa wartość CL), tym szybciej procesor uzyskuje dostęp do wymaganych, przechowywanych w pamięci danych. Uzyskanie wyższych częstotliwości taktowania odbywa się w kolejnych generacjach DDR kosztem wydłużenia czasów dostępu.

Same wartości CL nie powinny być porównywane wprost między różnymi generacjami, bo liczy się przede wszystkim rzeczywiste opóźnienie czasowe (w nanosekundach), które zależy zarówno od CL, jak i od częstotliwości zegara pamięci. Dlatego moduł o wyższym CL, ale pracujący z wyższym zegarem, może mieć podobne (albo nawet mniejsze) opóźnienie rzeczywiste niż moduł o niższym CL. W praktyce przy doborze pamięci znaczenie ma kompromis: przepustowość (MT/s) + opóźnienia (timingi) + to, co realnie obsługuje kontroler pamięci i płyta główna.

Pamiętajmy też, że procesor nie zawsze posiadał bezpośredniego połączenia z pamięcią RAM. W starszych platformach PC (szczególnie w klasycznych architekturach z FSB) kontroler pamięci bywał zintegrowany w mostku północnym płyty głównej, a procesor komunikował się z pamięcią „pośrednio”. We współczesnych platformach jest inaczej: typowym rozwiązaniem jest zintegrowanie kontrolera pamięci w samym procesorze, a komunikacja z chipsetem realizuje inne łącze niż historyczne FSB. Intel opisał tę zmianę m.in. przy okazji mikroarchitektury Nehalem, wskazując na zintegrowany kontroler pamięci i szybki interkonekt platformowy (Intel QuickPath Architecture). [Źródło: Intel, „Next Generation Intel® Microarchitecture (Nehalem)” (PDF): https://www.intel.com/pressroom/archive/reference/whitepaper_Nehalem.pdf]

Pamięci współczesne

W komputerach osobistych i przenośnych współczesnymi standardami pamięci operacyjnej są przede wszystkim DDR4 oraz DDR5 (w zależności od platformy). Z punktu widzenia użytkownika różnice sprowadzają się głównie do większej przepustowości, lepszej efektywności energetycznej oraz zmian w samej organizacji modułów i sposobie zasilania. Typowe napięcie zasilania komponentów DDR4 to 1,2 V, natomiast dla DDR5 obniżono je do 1,1 V, co pomaga ograniczyć zużycie energii. [Źródło: Kingston, „DDR5 Memory Standard: An introduction…”: https://www.kingston.com/en/blog/pc-performance/ddr5-overview]

W DDR5 wprowadzono też istotną zmianę w architekturze zasilania: część regulacji napięć, która wcześniej była realizowana na płycie głównej, została przeniesiona na sam moduł pamięci w postaci układu PMIC (Power Management IC). Dzięki temu zasilanie może być stabilizowane „bliżej” kości pamięci, co ułatwia pracę przy wyższych szybkościach i pomaga w kontroli jakości sygnału. [Źródło: Rambus, „New Power Architecture for DDR5 DIMMs” (29 Jul 2024): https://www.rambus.com/blogs/get-ready-for-ddr5-dimm-chipsets/]

Przy doborze pamięci do komputera kluczowa jest przede wszystkim zgodność z platformą sprzętową. Płyta główna oraz procesor muszą obsługiwać daną generację pamięci (DDR4 albo DDR5) – nie ma tu dowolności, ponieważ różnią się one elektrycznie i mechanicznie. Przykładowo, procesor Intel Core 12. generacji może współpracować zarówno z DDR4, jak i DDR5, ale konkretna płyta główna obsługuje tylko jeden z tych standardów. Z kolei starsze procesory lub tańsze platformy biurowe często w ogóle nie oferują wsparcia dla DDR5, więc wybór „nowszej” pamięci nie tylko nie poprawi wydajności, ale będzie po prostu niemożliwy do zrealizowania.

Drugim bardzo istotnym aspektem jest konfiguracja kanałów pamięci, czyli sposób, w jaki moduły RAM są podłączone do kontrolera pamięci w procesorze. W typowych komputerach osobistych spotykamy konfigurację Dual-Channel, co oznacza, że dwa identyczne moduły pamięci pracują równolegle, podwajając efektywną szerokość magistrali danych. W praktyce oznacza to wyraźny wzrost wydajności – szczególnie w grach oraz w aplikacjach intensywnie korzystających z pamięci. Przykładowo, zestaw 2×8 GB RAM w trybie Dual-Channel niemal zawsze będzie szybszy niż pojedynczy moduł 1×16 GB, mimo tej samej łącznej pojemności. W platformach profesjonalnych (workstation, serwery) spotyka się także więcej kanałów pamięci (np. Quad-Channel), co dodatkowo zwiększa przepustowość, ale wymaga odpowiednich procesorów i płyt głównych.

Na końcu warto spojrzeć na parametry samych modułów, takie jak efektywna szybkość transmisji (MT/s) oraz timingi, w szczególności opóźnienie CAS (CL). Wbrew pozorom wybór pamięci o najwyższej dostępnej częstotliwości nie zawsze ma sens. Przykładowo, w komputerze do pracy biurowej lub przeglądania internetu różnica między DDR4-3200 a DDR4-3600 będzie w praktyce niezauważalna. Z kolei w zadaniach takich jak kompilacja dużych projektów, obróbka wideo czy praca na maszynach wirtualnych większe znaczenie może mieć pojemność pamięci niż jej nominalna szybkość. Dlatego najrozsądniejsze podejście polega na dobraniu pamięci zgodnej z możliwościami kontrolera pamięci w procesorze oraz charakterem planowanych obciążeń, zamiast kierowania się wyłącznie „najwyższym numerem” w specyfikacji.

Warto też pamiętać, że pamięć operacyjna jest tylko jednym elementem całej drogi danych. Ostateczną wydajność systemu ogranicza często coś innego: sposób dostępu aplikacji do danych, rozmiar i organizacja pamięci podręcznych procesora, a także wydajność subsystemu dyskowego (SSD/HDD) oraz to, czy system operacyjny musi intensywnie korzystać z pliku wymiany.

1.3. Karty graficzne

Podstawowym zadaniem karty graficznej jest przetwarzanie danych graficznych oraz przygotowanie obrazu do wyświetlenia na ekranie monitora. Współczesne karty graficzne realizują te zadania w postaci wyspecjalizowanych, silnie równoległych obliczeń, a ich rola nie ogranicza się jedynie do przechowywania obrazu w pamięci karty i sterowania monitorem. Pierwsze karty graficzne potrafiły jedynie wyświetlać znaki alfabetu łacińskiego ze zdefiniowanego w pamięci karty generatora znaków - tryb tekstowy. Kolejna generacja kart graficznych potrafiła już wyświetlać w odpowiednim kolorze poszczególne punkty (piksele) - tryb graficzny. Nowoczesne procesory graficzne udostępniają wiele funkcji ułatwiających i przyśpieszających pracę programów. Możliwe jest narysowanie odcinka, trójkąta, wieloboku, wypełnienie ich zadanym kolorem lub wzorem, tzw. akceleracja 2D. Większość kart na rynku posiada również wbudowane funkcje ułatwiające tworzenie obrazu przestrzeni trójwymiarowej, tzw. akceleracja 3D. Niektóre posiadają zaawansowane algorytmy potrafiące na przykład wybrać tylko widoczne na ekranie elementy z przestrzeni. Rozróżnienie na tzw. akcelerację 2D i 3D miało znaczenie historyczne; we współczesnych kartach graficznych zarówno grafika dwu-, jak i trójwymiarowa realizowana jest przez ten sam potok przetwarzania. Źródło: NVIDIA Technologies and GPU Architectures: https://www.nvidia.com/en-us/technologies/ ; OpenGL Rendering Pipeline Overview (Khronos): https://wikis.khronos.org/opengl/Rendering_Pipeline_Overview

Przegląd kart graficznych rozpoczniemy od rozwiązań w zasadzie już historycznych:

MDA

Prekursorem kart graficznych, instalowanych w komputerach rodziny PC, był sterownik, który pojawił się w komputerach IBM PC w roku 1981, o nazwie **MDA** (ang. *Monochrome Display Adapter*). Karta pracowała tylko w trybie tekstowym o rozdzielczości 25 linii x 80 znaków [2]. Wyposażona w 4KB pamięci oferowała częstotliwość odchylania pionowego 50 Hz.

CGA

Następcą sterownika MDA była karta **CGA** (ang. *Color Graphics Adapter*) - opracowana przez IBM w 1982 roku karta jako pierwsza oferowała możliwość korzystania z trybu graficznego. Były to jednak tylko dwie rozdzielczości: 640x200 oraz 320x200 punktów. Karta oferowała 16 KB pamięci - większa rozdzielczość wyświetlana była jedynie w trybie monochromatycznym, zaś niższa "aż" w 4 kolorach. Tryb tekstowy możliwy był również w dwóch wariantach: 80 znaków x 25 linii, bądź 40 znaków x 25 linii, niestety matryca znaku miała rozmiary 8x8 pikseli. Karta oferowała maksymalną częstotliwość odświeżania pionowego 60 Hz.

Karta wykorzystywała spakowaną (ang. *packed*) metodę odwzorowania pamięci - w danym bloku pamięci RAM każdemu pikselowi obrazu odpowiadał fragment bajtu, zawierający numer koloru tego punktu (np. 1 bit - 2 kolory, 2 bity - 4 kolory, itd.).

Herkules

Karta Hercules pojawiła się w tym samym czasie co karta CGA. Była ona wyposażona dodatkowo w złącze równoległe, umożliwiające podłączenie drukarki.

Karta oferowała możliwość pracy w rozdzielczości 720x348 punktów (zarówno w trybie tekstowym, jak i graficznym), ale jedynie w trybie monochromatycznym. Wyposażona była w 64 KB pamięci. Znaki w trybie tekstowym wyświetlane były na podstawie matrycy 9x14 punktów. Karta nie była zgodna z BIOS-em IBM PC, gdyż nie implementowała standardów CGA, co wymagało stosowania dedykowanych sterowników.

Aby umożliwić szybszy dostęp do danych pamięć została podzielona była na dwie strony graficzne, natomiast każda ze stron - na cztery banki.

EGA

Karta EGA (ang. *Enchanced Graphics Adapter*) to kolejny etap rozwoju CGA. Karta oferowała wyświetlanie obrazu w rozdzielczości 640x350 punktów przy 16 kolorach (wybieranych z palety 64 kolorów). Zaopatrzona była w 256 KB pamięci. Rozdzielczość w trybie tekstowym wynosiła 80x43, przy matrycy znaku 8x14.

Sterownik EGA składał się z czterech głównych bloków funkcjonalnych:

- *Układ sekwencyjny* - generuje sygnał zegarowy; przesyła dane pomiędzy pamięcią obrazu, układem graficznym i układem określania atrybutu; odpowiada za wybór lokalizacji wyświetlanych znaków.
- *Układ graficzny* - przekazuje dane pomiędzy pamięcią obrazu, układem graficznym i układem określania atrybutu.
- *Układ sterowania atrybutem* - służy do zmiany kolorów zapisanych w pamięci obrazu na indeksy kolorów zdefiniowanych w rejestrach wzorców kolorów.
- *Układ sterowania wyświetlaczem* - odpowiada za zachowanie zależności czasowych podczas wyświetlania obrazu oraz wyświetla kursor.

Pamięć wideo opisywanej karty podzielona jest na cztery 64KB obszary (rozwiązanie to jest wykorzystywane również w kartach VGA). Cztery obszary pamięci zawierają bity indeksu koloru piksela, który następnie jest mapowany na rzeczywisty kolor z palety. Tak więc jeden piksel zawierał swoje składowe w czterech blokach pamięci. Dzięki takiej strukturze 256 KB zajmuje 64 KB przestrzeni adresowej. Rozwiązanie to nosi nazwę metody płatowej (ang. *planar, bit maped*), a jego wadą jest to, że utrudniony zostaje dostęp do danych.

VGA

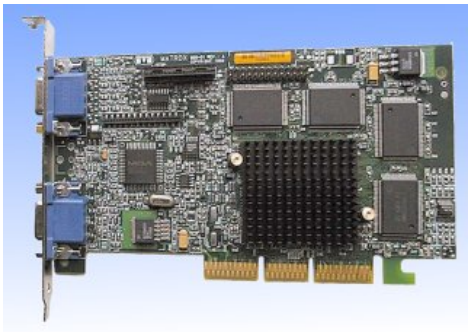
Karta VGA (ang. *Video Graphics Array*) to kolejny standard firmy IBM, opracowany z myślą o aplikacjach graficznych. Sterownik ten jest w stanie emulować wszystkie dotychczas opisane standardy.

Cechą wyróżniającą kartę VGA jest fakt, że wysyłała ona do monitora sygnały analogowe (poprzedniczki operowały na sygnałach cyfrowych), dzięki czemu zwiększono ilość wyświetlanych kolorów. Zajmuje się tym wyspecjalizowany układ przetwornika cyfrowo-analogowego (DAC - ang. *Digital Analog Converter*), który jest w stanie przedstawić każdą z barw w postaci określonej liczby poziomów (np. 64). Standardowy sterownik VGA umożliwiał wyświetlanie 25 wierszy znaków w 80 kolumnach (matryca znaku 9x16). Znak może być wyświetlany w jednym z 16 kolorów, natomiast kolor tła dla każdego znaku może być inny.

W trybie graficznym karta VGA umożliwia wyświetlenie obrazu między innymi o rozmiarach 640x480 punktów przy 16 kolorach (wybranych z palety 256 kolorów). Maksymalną liczbę kolorów - 256 - osiągnąć można przy rozdzielczości 320x200 punktów.

SVGA

Karty SVGA (ang. *Super VGA*) są rozszerzeniem techniki VGA. Sterowniki SVGA wykorzystują tzw. technikę stronicowania, polegającą na kojarzeniu z niewielkim obszarem pamięci (oknem), w przestrzeni adresowej, różnych fragmentów większego obszaru pamięci (stron lub banków). Tak więc, zapis/odczyt adresu położonego wewnątrz okna, powoduje zapis/odczyt odpowiadającego mu bajtu w banku. Rozmiar banku i okna wynosi zwykle 64 KB. Aby dostać się do pamięci spoza bieżącego banku, należy zmienić zawartość rejestru sterującego położeniem banku (ang. *Bank Start Adress*). Wszystko to po to, aby efektywniej wykorzystać 64 KB okna pamięci w przestrzeni adresowej, którą ma zarezerwowaną procesor na pamięć obrazu.



Karta graficzna SVGA firmy Matrox

Dzięki takim, i wielu innym innowacjom, możliwe jest korzystanie z dużej pojemnościowo pamięci, co umożliwiało kartom graficznym osiąganie rozdzielczości 1280x1024 i wyższych, przy palecie kolorów 16.7 mln. (true color).

Rozwiązania współczesne

Współczesna karta graficzna to wysoko-wydajnościowy system mikroprocesorowy, zwykle składający się z pięciu komponentów: interfejsu systemowego, pamięci graficznej, procesora graficznego GPU (Graphics Processing Unit), bufora ramki oraz układów wyjścia obrazu, które w przypadku analogowego standardu VGA realizowały funkcję RAMDAC (Random Access Digital/Analog Converter). Źródło: Intel Graphics for Linux - Developer Reference Overview: https://www.intel.com/content/www/us/en/docs/graphics-for-linux/developer-reference/overview.html

Interfejs systemowy znajduje się najbliżej płyty głównej. Obecnie stosuje się złącze PCI Express. Za jego pośrednictwem nieobrobione dane są ładowane do pamięci graficznej, w której przechowywane są wszelkie obiekty graficzne i tekstury. Jej pojemność waha się od kilku do kilkudziesięciu gigabajtów, w zależności od klasy karty. Informacje są następnie odczytywane przez procesor graficzny, który przygotowuje do wyświetlenia na ekranie pozycje, ruchy i fakturę wszelkich obiektów widocznych w trójwymiarowej scenie. Po obróbce gotowy obraz jest umieszczany w buforze klatek i wędruje do RAMDAC – układu, który przetwarza cyfrowe obrazy, nadając im formę analogową, odpowiednią dla wyjścia VGA, lub przesyła je do cyfrowych wyjść DVI, HDMI lub DisplayPort. W przypadku większości wyjść cyfrowych istnieje możliwość prostej konwersji sygnału poprzez zakupienie odpowiedniego kabla. Wyjście DVI jest wyjściem mieszanym - to znaczy oprócz sygnałów cyfrowych istnieje także wyprowadzenie sygnałów analogowych, dzięki czemu po zastosowaniu prostej przejściówki można kartę podłączyć do odbiornika obsługującego analogowy standard VGA. Źródło: NVIDIA Technologies and GPU Architectures: https://www.nvidia.com/en-us/technologies/



Współczesny procesor graficzny - jak widzicie potrzebuje on chłodzenia równie wydajnego jak procesor główny

Centralnym elementem każdej karty graficznej jest procesor graficzny (ang. Graphics Processing Unit, GPU). Jako pierwsza tego terminu użyła firma NVIDIA wprowadzając na rynek karty graficzne GeForce 256. To przełomowe wydarzenie dla akceleracji grafiki w systemach domowych miało miejsce 31 sierpnia 1999 roku, wcześniej systemy takie były dostarczane wyłącznie jako specjalizowane karty profesjonalne. Głównym zadaniem GPU było wykonywanie obliczeń potrzebnych do uzyskania akcelerowanej grafiki 3D, co spowodowało odciążenie procesora CPU z konieczności wykonywania tego zadania. W tej sytuacji mógł on zająć się innymi obliczeniami, co skutkowało zwiększeniem wydajności komputera podczas renderowania grafiki. Nowoczesne procesory graficzne wyposażone są w szereg instrukcji, których nie posiada procesor komputera, i w całości są wielkimi procesorami typu SIMD. Ponadto są wyposażone w dodatkowe bloki specjalizowane, np do kompresji / dekompresji obrazu. W tym przypadku rola procesora w komputerze ograniczona jest do dostarczenia danych, a pracochłonnym dekodowaniem i wyświetleniem filmu zajmuje się procesor karty graficznej. Współczesne układy GPU radzą sobie także z konwersją wideo, czyli zmianą formatu filmu. Prawdziwym wyzwaniem dla współczesnych kart są jednak wymagające gry komputerowe.

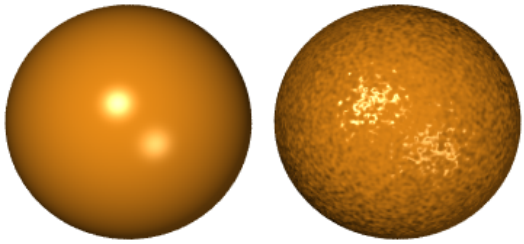
W przypadku gier zadania karty składają się na tzw renderowanie obrazu, czyli przekształcenie wektorowego, uproszczonego opisu sceny na jej obraz widziany z konkretnego punktu. Zazwyczaj te zadanie jest wykonywane dwuetapowo:

- W pierwszym etapie wykorzystywane są tzw. shadery, w grafice trójwymiarowej odpowiadające za cieniowanie obiektów. Shader pozwala na dużo bardziej skomplikowane modelowanie oświetlenia i materiału na obiekcie niż standardowe modele oświetlenia i teksturowanie. Jest jednak dużo bardziej wymagający obliczeniowo i dlatego dopiero od kilku lat sprzętowa obsługa shaderów jest obecna w kartach graficznych dla komputerów domowych. Shader jest programem wykonywanym równolegle na wielu jednostkach wykonawczych procesora graficznego. Każda karta graficzna ma kilka, kilkanaście, na nawet kilkaset shaderów w różnych wariantach. Jeśli monitor ma na przykład wyświetlić pędzący samochód wyścigowy, to odpowiada za to Vertex Shader, który błyskawicznie oblicza nową pozycję pojazdu w każdym obrazie i dodatkowo dba o właściwe oświetlenie obiektu, kiedy w scenie występuje wiele źródeł światła. Pixel Shader z kolei odpowiada za zmiany powierzchni i koloru obiektu. Kiedy w danej scenie samochód szybko się zbliża, uaktywnia się natomiast Geometry Shader, który błyskawicznie zaopatruje pojazd w nowe krawędzie i narożniki. W nowszych architekturach rolę tę często przejmują mechanizmy teselacji oraz mesh shaders. O ile samochód jest wyświetlany jako obiekt o niewielkich rozmiarach, daleko w tle, to do jego wyświetlenia wystarczy praktycznie sam obrys. Jeśli jednak jego obraz wypełnia cały ekran, to realistyczny wygląd może mu nadać tylko mnóstwo drobnych szczegółów. W stosunku do standardowych modeli oświetlenia, stosowanych do generowania grafiki w czasie rzeczywistym, shadery dają możliwość uwzględnienia między innymi:
  - refrakcji
  - odbić lustrzanych
  - mapy przemieszczeń

  Źródło: OpenGL Rendering Pipeline Overview (Khronos): https://wikis.khronos.org/opengl/Rendering_Pipeline_Overview
- kiedy shadery wykonają swoją pracę, to obiekt musi zostać "oklejony" kolorami i tak zwanymi teksturami, za co odpowiada jednostka teksturująca. Procesor graficzny zapisuje w pamięci karty graficznej możliwie najbardziej kompletne tekstury już na początku sceny.

Do najważniejszych funkcji współczesnych akceleratorów graficznych można zaliczyć:

- **Anisotropic Filtering** (Filtrowanie anizotropowe) - jest to technika wyostrzania tekstur (czyli rysunków jakimi pokryte są ściany obiektów) w trójwymiarowej grafice komputerowej, które znajdują się w dalszej odległości od kamery (lub postaci sterowanej przez gracza).
- **Bump Mapping** (mapowanie wypukłości) – w grafice 3D technika teksturowania, która symuluje niewielkie wypukłości powierzchni, bez ingerencji w geometrię obiektu trójwymiarowego.



Kula prezentowana bez (po lewo) i z (po prawo) mapowaniem wypukłości

- **HDR** (High Dynamic Range) - technologia generowania sceny w grafice trójwymiarowej, której efektem jest renderowanie świata z realistycznym oświetleniem. Największa różnica dostrzegana jest w bardzo ciemnych lub bardzo jasnych fragmentach sceny, gdzie symulowane jest natężenie światła wykraczające poza zakres możliwy do osiągnięcia na ekranie monitora (np. efekt oślepienia po spojrzeniu na słońce).
- **Shader**

Poza tym karty graficzne oferują inne sprzętowe efekty, jak mgła, przezroczystość (dodatkowy kanał Alpha) oraz szereg technik poprawy jakości obrazu. Jednak znaczenie współczesnych procesorów graficznych daleko wykracza poza klasyczne renderowanie grafiki 3D.

Możliwości GPU nie ograniczają się jedynie do grafiki trójwymiarowej. We współczesnych procesorach graficznych umieszczono tysiące prostych jednostek arytmetyczno-logicznych (ALU), zdolnych do wykonywania tego samego programu na wielu danych jednocześnie. Dzięki temu GPU doskonale nadają się do obliczeń silnie równoległych. W wielu zastosowaniach, takich jak symulacje naukowe, analiza danych czy obliczenia finansowe, karty graficzne osiągają wydajność rzędu kilkudziesięciu, a nawet ponad stu razy większą niż procesory CPU. Prowadzi to do wykorzystania GPU jako procesorów ogólnego przeznaczenia, określanych mianem GPGPU (General Purpose Graphics Processing Unit).

Prekursorem upowszechnienia obliczeń GPGPU była firma NVIDIA, która wprowadziła platformę CUDA (Compute Unified Device Architecture), obejmującą model programowania, kompilatory oraz biblioteki dla języków C i C++. CUDA umożliwia tworzenie aplikacji dzielonych na tysiące lekkich wątków wykonywanych równolegle przez jednostki GPU. Równolegle rozwijane są rozwiązania niezależne od jednego producenta, takie jak standard OpenCL, utrzymywany przez organizację Khronos Group. Firma AMD rozwija natomiast własną otwartą platformę ROCm (Radeon Open Compute), której kluczowym elementem jest język i warstwa kompatybilności HIP (Heterogeneous-Compute Interface for Portability), pozwalająca tworzyć aplikacje przenośne pomiędzy architekturami AMD i NVIDIA. Źródła: NVIDIA CUDA Programming Guide: https://docs.nvidia.com/cuda/ ; AMD ROCm Documentation: https://rocmdocs.amd.com/ ; Khronos OpenCL Registry: https://registry.khronos.org/OpenCL/

Jednym z najważniejszych współczesnych obszarów zastosowań GPU jest sztuczna inteligencja, w szczególności uczenie maszynowe i głębokie sieci neuronowe. Trening modeli AI polega na wykonywaniu ogromnej liczby operacji macierzowych, takich jak mnożenie macierzy i wektorów, które idealnie pasują do architektury masowo równoległej GPU. Z tego powodu zarówno centra danych, jak i stacje robocze do uczenia modeli wykorzystują dziś przede wszystkim akceleratory graficzne, często wyposażone w wyspecjalizowane jednostki obliczeniowe, takie jak Tensor Cores w kartach NVIDIA czy Matrix Cores w architekturach AMD.

Procesory graficzne są również szeroko stosowane na etapie wnioskowania (inference), czyli wykorzystania wytrenowanych modeli AI w praktycznych zastosowaniach. Dotyczy to m.in. rozpoznawania obrazu i mowy, analizy strumieni wideo, autonomicznych pojazdów, systemów rekomendacyjnych oraz przetwarzania języka naturalnego. Dzięki wysokiej przepustowości pamięci i równoległemu przetwarzaniu danych GPU umożliwiają realizację takich zadań w czasie zbliżonym do rzeczywistego, co czyni je kluczowym elementem współczesnych systemów opartych na sztucznej inteligencji. Źródła: NVIDIA Deep Learning Performance: https://developer.nvidia.com/deep-learning-performance ; AMD AI and Machine Learning Overview: https://www.amd.com/en/solutions/ai

W testach syntetycznych okazuje się, że wydajność GPU, dzięki masowej równoległości przetwarzania (tysiące programowalnych jednostek wykonawczych w jednym procesorze graficznym pracuje równolegle), przewyższa o rząd wielkości wydajność najlepszych procesorów CPU w zadaniach podatnych na równoległość.

Aktualnie na rynku liczą się przede wszystkim producenci procesorów graficznych - nVidia, AMD oraz - dla rozwiązań mniej wymagających - Intel. Znów - podobnie jak w przypadku procesorów - opis konkretnych nowych rozwiązań w podręczniku jest z góry skazany na permanentną nieaktualność, więc po informacje o najnowszych rozwiązaniach odsyłamy Was do internetu. Źródło: Intel Arc Graphics Overview: https://www.intel.com/content/www/us/en/products/details/discrete-gpus/arc.html

1.4. Pamięci masowe

Dyski twarde we współczesnych systemach

Klasyczne dyski twarde HDD, mimo dynamicznego rozwoju pamięci półprzewodnikowych, wciąż znajdują zastosowanie we współczesnych systemach komputerowych. Ich główną zaletą pozostaje bardzo korzystny stosunek ceny do pojemności, dzięki czemu są powszechnie wykorzystywane jako nośniki danych archiwalnych, kopii zapasowych, materiałów multimedialnych oraz w systemach NAS i serwerach plików.

W praktyce dyski HDD nie nadają się dziś do roli nośnika systemowego w komputerach osobistych – ich wysoki czas dostępu oraz niska wydajność przy operacjach losowych powodują zauważalne opóźnienia w pracy systemu. Typowym i rozsądnym rozwiązaniem jest stosowanie konfiguracji hybrydowej: szybki dysk SSD jako nośnik systemu operacyjnego i aplikacji oraz pojemny HDD do przechowywania dużych zbiorów danych.

Warto także podkreślić, że wydajność dysku HDD ma znaczenie głównie przy odczycie i zapisie sekwencyjnym dużych plików. W zastosowaniach takich jak monitoring, archiwizacja nagrań wideo czy magazyn danych multimedialnych, HDD nadal spełniają swoją rolę w sposób ekonomicznie uzasadniony.

Dyski SSD – podstawowy nośnik danych we współczesnych komputerach

Dyski SSD stały się obecnie podstawowym nośnikiem danych w komputerach osobistych i przenośnych. Ich kluczową przewagą nad dyskami HDD jest bardzo niski czas dostępu oraz wysoka wydajność przy operacjach losowych, co bezpośrednio przekłada się na szybkość uruchamiania systemu operacyjnego, aplikacji oraz ogólną responsywność komputera.

Współczesne dyski SSD występują w dwóch głównych odmianach: podłączane przez interfejs SATA oraz korzystające z protokołu NVMe i magistrali PCI Express. Dyski SSD SATA osiągają przepustowość ograniczoną przez interfejs (około 550 MB/s), natomiast nośniki NVMe mogą oferować transfery rzędu kilku tysięcy MB/s, co ma szczególne znaczenie w zastosowaniach profesjonalnych.

Należy odróżnić format fizyczny M.2 od protokołu komunikacyjnego NVMe. Nośnik M.2 może wykorzystywać zarówno interfejs SATA, jak i PCI Express. Dopiero połączenie formatu M.2 z protokołem NVMe pozwala na pełne wykorzystanie możliwości magistrali PCI Express.

Pamięci flash i trwałość nośników SSD

Współczesne dyski SSD wykorzystują różne typy komórek pamięci flash: SLC, MLC, TLC oraz QLC. Im więcej bitów przechowywanych jest w jednej komórce, tym większa pojemność i niższy koszt produkcji, ale jednocześnie mniejsza trwałość oraz wydajność zapisu. Z tego powodu nośniki konsumenckie najczęściej bazują na pamięciach TLC lub QLC, wspieranych przez mechanizmy buforowania SLC.

Trwałość dysków SSD określana jest parametrem TBW (Total Bytes Written), czyli maksymalną ilością danych, jaką można zapisać na nośniku w całym okresie jego eksploatacji. W praktyce, przy typowym użytkowaniu domowym lub biurowym, współczesne dyski SSD osiągają żywotność liczonych w wielu latach, a często przekraczającą okres ich realnego użytkowania.

Typowe konfiguracje pamięci masowych

W komputerach biurowych i edukacyjnych najczęściej spotyka się pojedynczy dysk SSD o pojemności od 256 do 512 GB. W komputerach do gier oraz zastosowań profesjonalnych popularne są konfiguracje z szybkim dyskiem NVMe na system i aplikacje oraz dodatkowym nośnikiem HDD lub SSD SATA na dane. W laptopach i ultrabookach standardem stał się pojedynczy nośnik SSD NVMe.

Informacje techniczne o poszczególnych typach pamięci

Dyski twarde

Dyski twarde HDD (ang. *Hard Disk Drive*) zostały tak nazwane z powodu swej sztywnej konstrukcji. Dyski twarde nie zawsze były takie "twarde". Kiedyś, przed przenoszeniem dysku z miejsca na miejsce, trzeba było zaparkować głowice, czyli uruchomić specjalny program, który zajmował się przemieszczeniem głowic poza obszar magnetyczny dysku.



Dysk twardy firmy Quantum

Dzisiaj dyski operacje takie wykonują automatycznie, ponadto są bardzo odporne na wstrząsy. Dyski twarde zawierają w swej obudowie kilka, a nawet kilkanaście talerzy (standardowo 3 talerze magnetyczne). Talerze wirują prędkością 3600-7200 obrotów na minutę (ang. RPM - *Rounds Per Minute*), a niektóre dyski SCSI kręcą się z prędkością 15 000 RPM (250 obrotów na sekundę). Wewnątrz pyłoszczelnej obudowy dysku twardego znajdują się (oprócz głowicy i talerzy): układy sterowania silnikiem napędu dysków, silnikiem przesuwu głowic (służącym do pozycjonowania) oraz głowicami zapisu/odczytu, a także inne układy sterowania i kontroli. Dzięki dużej prędkości w ruchu obrotowym wytwarza się poduszka powietrzna pod głowicą zapisu/odczytu, dlatego łatwo może ona być utrzymywana w stałej odległości od talerza (głowica nie dotyka dysku podczas pracy !). Dzięki dużej prędkości obrotowej możliwe jest również uzyskiwanie dużych prędkości transmisji danych.

Najważniejszymi parametrami dysków twardych są:

- szybkość transmisji (transfer) danych;
- prędkość obrotowa (5400 RPM, 7200 RPM, 15 000 RPM);
- Średni czas dostępu (ang. *average access time*) wyrażany w ms (np. 10ms). Na tę wielkość składają się:
    - średni czas wymagany do umieszczenia głowic nad odpowiednim cylindrem (ang. *average seek time*);
    - opóźnienie rotacyjne związane z umieszczeniem głowicy nad wybranym sektorem;
- pojemność (na dzień dzisiejszy sięgająca terabajtów danych);

Należy tu zauważyć, że prędkość dysku zależy także od rodzaju złącza (interfejsu).

Na dzień dzisiejszy standardów jest kilka, przy czym najważniejsze z nich to:

SATA, SATA-II, SATA-III – to obowiązujący od kilku lat standard w przypadku komputerów osobistych. SATA (ang. *Serial Advanced Technology Attachment*) jest szeregowym interfejsem umożlwiającym podłączanie różnorakich urządzeń zewnętrznych. Pierwsza wersja oferuje przepustowość do 1,5 Gbit/s (ok 179 MB/s), druga 3 GBit/s, trzecia 6 GBit/s. Ponadto w tym standardzie istnieje także kilka typów złącz. Oprócz klasycznego wtyku 7-pinowego, możliwe jest stosowanie standardu eSATA do podłączania dysków na zewnątrz obudowy komputera, przy ograniczeniu długości kabla do ok. 2 metrów. Rozwinięcie standardu eSATA to xSATA - pozwala na stosowanie kabli do 8 metrów długości. Inne zastosowanie ma złącze mSATA, opracowane głównie w celu miniaturyzacji samej wtyczki, i wykorzystywane w rozwiązaniach przenośnych (małe dyski, netbook-i, itp).

SAS – Serial Attached SCSI – następca standardu SCSI. Używany głównie w dyskach serwerowych. Fizycznie SAS jest kompatybilny z SATA. Dyski SATA można podłączać do kontrolera SAS, ale zgodności w drugą stronę już nie ma. Osiągane prędkości transferu to 12 Gbit / s.

W praktyce, we współczesnych komputerach osobistych spotyka się głównie dyski HDD o prędkości 5400 lub 7200 RPM (w laptopach częściej 5400 RPM, w komputerach stacjonarnych częściej 7200 RPM). Wydajność takich dysków zależy nie tylko od obrotów, ale też od gęstości zapisu oraz wielkości bufora (cache) – typowe wartości cache to dziś 64–256 MB. W zastosowaniach domowych dyski HDD osiągają zwykle transfer sekwencyjny rzędu około 100–250 MB/s (zależnie od modelu i pojemności), natomiast w operacjach losowych ich wydajność jest wielokrotnie niższa niż w przypadku SSD, ponieważ głowica musi wykonywać ruch mechaniczny.

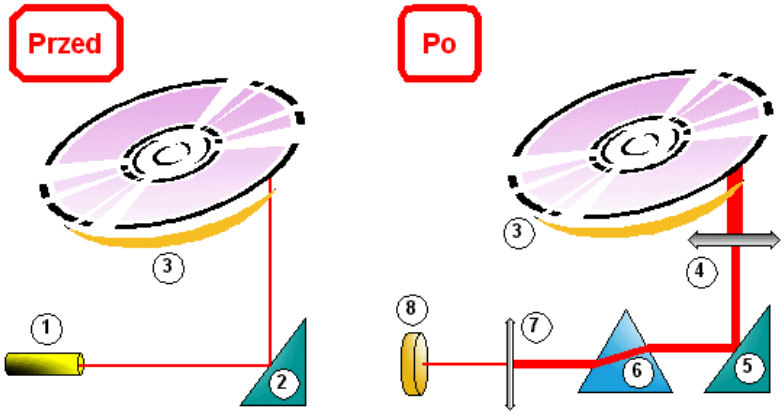
Przy doborze dysku HDD warto zwrócić uwagę na jego przeznaczenie: inne modele projektuje się do pracy ciągłej w serwerach/NAS (często z firmware i parametrami zoptymalizowanymi pod RAID), a inne do sporadycznego obciążenia w komputerze domowym. W praktyce oznacza to różnice w kulturze pracy, poborze mocy oraz odporności na wibracje - rozwiązania domowe są tanie i energooszczędne, ale zwykle mniej odporne.

Napędy optyczne

Prace nad napędem CD-ROM rozpoczęte zostały w roku 1978. Pierwszy napęd powstał dzięki firmie Phillips, ale odtwarzacz CD-ROM, jaki znamy w dzisiejszej postaci zaprojektowany został przez team (ang. "team") firm Phillips i Sony. W połowie lat 80 minionego wieku, napędy te weszły do szerszej dystrybucji, zaś lata 90 przyniosły im niezwykłą popularność głównie dzięki stosunkowo niskiej cenie oraz dużej pojemności jaką oferowały płyty CD-ROM.

Zasada działania napędu CD-ROM jest prosta (patrz rysunek poniżej). Otóż:

1. *Dioda laserowa* (1) emituje na lustro (2) promień podczerwony o niskiej energii.
2. *Silnik servo*, przesuwając lustro (2) na rozkaz mikroprocesora, ustawia promień na właściwej ścieżce płyty (3).



Zasada działania napędu CD-ROM

3. Po odbiciu promienia przez płytę (3), jego światło jest zbierane i skupiane przez pierwszą soczewkę (4) znajdującą się pod płytą, odbijane od lustra (5) i kierowane do układu pryzmatów (6).
4. *Układ pryzmatów* (6) kieruje promień na następną soczewkę skupiającą (7).
5. Druga soczewka kieruje promień światła na *fotodetektor* (8), który zamienia go na ciąg impulsów elektrycznych.
6. Impulsy elektryczne są dekodowane przez mikroprocesor i wysyłane do komputera.

Zagłębienia wytrawione na płycie CD-ROM (ang. *pit*) sprawiają, że światło odbite od pit'a różni się intensywnością (energią) od tego, które na pit'a nie trafiło. Zmiana energii wiązki lasera traktowana jest więc jako pojawienie się informacji i przekształcana przez fotodetektor na impulsy elektryczne.

Ponieważ przekłamanie informacji o pojedynczym bicie może być groźne np. dla działania programu typu *.exe, stosuje się zaawansowane techniki wykrywania i korekcji błędów. Dzięki nim prawdopodobieństwo niewykrycia błędu jest rzędu 10^(-25). Sama korekcja wymaga zaś 288 B na każde 2048 B danych.

Najważniejszymi parametrami napędów CD-ROM są :

- *Szybkość transmisji* - określa jak szybko może odbywać się komunikacja pomiędzy jednostką centralną a urządzeniem. Dla napędów CD-ROM przyjęło się używać oznaczeń 1x, 2x, 8x, itp. Są to oznaczenia właśnie szybkości transmisji. Symbol 1x określa przepływ danych przez 75 sektorów danych po 2048 B w ciągu sekundy (format CD), co daje szybkość 150 kB/s. Naturalnie sybole 2x, 4x, itd. są wielokrotnościami wymienionej prędkości.
- *Czas dostępu* (ang. *access time*) - opóźnienie pomiędzy momentem zlecenia odczytu, a chwilą uzyskania danych (typowa wartość 100 ms). Należy pamiętać, że inny jest czas dostępu do danych położonych blisko środka, a inny przy brzegu płyty.

Napęd CD-ROM można podłączyć do komputera korzystając z interfejsów, z których najpopularniejsze to SCSI/ASPI oraz IDE/ATAPI.

DVD

Skrót DVD *Digital Versatile Disk* - dysk zawierający różne dane. W domyślnym znaczeniu chodzi o dane cyfrowe, przede wszystkim cyfrowo zapisane multimedia. Obecnie jest najpopularniejszym nośnikiem optycznym. Pojedynczy dysk DVD jest w stanie pomieścić nawet do 15GB, oczywiście jeśli to będzie płytka dwuwarstwowa dwustronna. Najczęściej jednak dysk ten jest jednowarstwowy jednostronny o pojemności 4.7GB. Podobnie jak w przypadku płytki CD podawane są pojemności w minutach - w takim przypadku pojemność obliczana jest dla filmu i ścieżki dźwiękowej o bardzo dobrej jakości - dla płytki jednowarstwowej (4.7GB) to 133 minuty. Wymiarami zewnętrznymi płytka DVD nie różni się od CD. Grubość 1.2 mm , średnica 120 mm (mini 80mm). Podobnie są również pod względem kolejności i układu warstw. Laser wykorzystywany do pracy w tym napędzie pracuje wykorzystując światło o długości 650nm

Cieńsza jest natomiast warstwa z poliwęglanu. Związane jest to z technologia wykonywania nośników dwustronnych. Są one składane „plecami” do siebie i wymiar grubości 1.2 mm zostaje przez to zachowany.

Formaty DVD

DVD-ROM - czyli płytka tylko do odczytu

Może ona zawierać różne dane, trzy najpopularniejsze formaty to DVD-ROM (dane komputerowe bez determinowania ich treści), DVD-VIDEO (materiał video w formacie akceptowalnym przez każdy odtwarzacz video) oraz DVD-AUDIO (najmniej popularny z wymienionych - zawiera muzykę o jakości wyższej niż CD)

DVD-R

Płytki jednorazowego zapisu. Wyróżnia się dwa, niezgodne ze sobą, formaty: -R oraz +R (na szczęście aktualnie praktycznie wszystkie napędy DVD-RW umożliwiają odczyt - zapis w nich obu). Pierwszy zapisywalny typ nośnika w technologii DVD. Stanowią odpowiednik płytki CD-R. Obecnie prędkość zapisu takich płyt osiąga wartość 18x

DVD-RW

Płytki wielokrotnego zapisu. Istnieją dwa formaty podobnie jak poprzednie. DVD-RW /+RW . Płytki takie mogą być nagrywane do ok. 1000 razy (oczywiście rozwiązania firmowe, i te zapisy muszą być dokonane w krótkim czasie).

DVD–RAM

Płytka ta po pierwszym formatowaniu służy jak dysk twardy. Dane odczytywane jak i zapisywane na niej są w czasie rzeczywistym. Standard ten popularny jest Japonii, w innych rejonach świata przegrał z konkurencyjnym DVD-RW. Cechuje go wysoka niezawodność i bezpieczeństwo danych. Trwałość tego medium szacowana jest na ok. 100 000 cykli. Pierwsza generacja jednostronnej płytki miała 2.58 GB potem doprowadzono ją do typowej pojemności 4.7 GB. Doskonale sprawdzają się tam gdzie stosowano do tej pory napędy taśmowe oraz napędy MO. Stosowany system plików to UDF lub FAT32. **Wwaga** - płytki DVD RAM nie da się odczytać w standardowym napędzie DVD

Nowością wprowadzoną przy uatrakcyjnianiu nagrywarek DVD jest technologia **LightScribe -** jest to możliwość tworzenia za pomocą lasera nagrywarki etykiet na dyskach CD i DVD. Obraz powstaje na górnej stronie płytki. Wadą tej technologii jest stosunkowo długi czas wypalania.



Typowa nagrywarka DVD z funkcją LightScribe

Praktyczne stosowanie technologii LightScribe wymaga również stosowania przeznaczonych do tego płytek i oprogramowania.

HD DVD

Pierwotna nazwa AOD (Advanced Optical Disk) - standard opracowany wspólnie przez NEC i Toshiba. Dysk HD DVD był bardzo zbliżony do poprzednika-DVD. Płytki są o średnicy 120mm składają się z dwóch warstw o grubości 0.6 mm każda. Płytki mogą być jednowarstwowe, dwuwarstwowe, jednostronne lub dwustronne. Zasadniczą różnicą jaka została wprowadzona to zastosowanie światła laserowego o długości 405nm - koloru niebieskiego, a mniejsza długość światła oznacza możliwość gęstszego upakowania danych (zmniejsza się rozmiar najmniejszego możliwego do odróżnienia punktu).Prosta zmiana długości światła pozwoliła uzyskać 15GB na jedną warstwę tłoczoną - dla warstwy nagrywanej do 20GB. Standard przewiduje stosowanie jednego systemu plików - UDF. Napędy HD-DVD zachowują wsteczną zgodność - większość z nich pozwala na odczyt płyt poprzednich generacji (DVD i CD). W format HD-DVD mogą być wbudowane zabezpieczenia przed kopiowaniem i nieautoryzowanym dostępem.

Standard ten nigdy nie zdobył wielkiej popularności

Blu-Ray

Nośnik danych oraz napęd nad którym prace prowadziły firmy Sony, Philips Hitachi, Sharp, Samsung, TDK. Konkurencyjna technologia wobec HD DVD. Płytki Blue Ray są w stanie pomieścić w jednej warstwie 25GB danych. Zatem dla płytki dwuwarstwowej to już 50GB danych. Osiągnięcie tak ogromniej pojemności na krążku wymagało wprowadzenia zupełnie nowej technologii. Skonstruowano układ optyczny który wypala zmniejszone pole (pit) w warstwie światłoczułej płytki. Płytka w stosunku do poprzednich rozwiązań charakteryzuje się następującymi cechami:

- zmniejszona grubość warstwy ochronnej do 0,1 mm . Przy czym została ona pokryta materiałem odpornym na zabrudzenia.
- zmieniona aperturę numeryczna soczewki do wartości 0,85.

Długość wiązki zastosowanego lasera ma wartość 405 nm. Jest to kolor niebieski- stąd to właśnie pochodzi nazwa całego standardu- Blue-ray. Ponadto moc lasera potrzebna do nagrania zmniejszona została o jedna czwartą w porównaniu do potrzebnej do nagrania w standardzie DVD.

Dyski BD postanowiono w związku ze zwiększoną wrażliwością na zabrudzenia np. z odcisków palców, pakować w kasety.

Napęd Blu-ray powszechnie zastosowano w SONY PlayStation 3 i 4.

Pamięci Flash podłączane pod port USB

Jest wiele różnych nazw do określenia tego rodzaju urządzenia. Najczęściej można spotkać się z określeniem:

- PenDdrive
- Memory Stick
- USB Flash Drive



Typowa pamieć pendrive

Wspólna cecha wszystkich urządzeń typu PenDrive jest obecność złącza USB. Ma dwojaką role, zasilania i interfejsu dla danych. Pamięci tego typu charakteryzują się dobrymi parametrami. Brakiem części ruchomych a przez to dużą trwałością urządzenia. Średni czas pracy obliczono na 10 lat. Ściślej zależy on od liczby dokonywanych zapisów i odczytów. Jedną z najważniejszych cech jest szybkość zapisu i szybkość odczytu. Nowoczesne pamięci mają średni transfer rzędu kilkudziesięciu MB/s. Przy czym należy zauważyć ze transfer ten zależy od jakości zastosowanej pamięci, wersji łącza USB aż po wielkości pliku kopiowanego. Urządzenia wyprodukowane na złączu USB Low Speed (USB v.1.0) to zaledwie 1.5Mb/s.


Przykład popularnego PenDrive'a

Przypadku Urządzeń z określeniem USB 2.0 Full Speed to 12 Mb/s. Następne są urządzenia z nazwa Hi-Speed - maksymalnie 480Mb/s. Przy kopiowaniu małej liczby plików prędkość kopiowania ma drugorzędne znaczenie. Im plik większy tym transfer kopiowania wyższy.

Pamięci PenDrive mają szerokie zastosowania. Od zwykłego zastępowania zwykłej dyskietki podczas przenoszenia plików z jednego komputera do innego, aż po wyrafinowane urządzenia mogące przechowywać dane w zaszyfrowanej postaci, czy też po napędy symulujące dysk twardy komputera. Stosowane są również w połączeniu z odtwarzaczami mp3, mp4.

Współcześnie kluczowym parametrem pendrive nie jest już sama „pojemność”, lecz standard interfejsu USB oraz realna wydajność zapisu. W sprzedaży spotyka się nośniki zgodne z USB 3.2 Gen 1 (dawniej USB 3.0), USB 3.2 Gen 2, a czasem także pamięci z wtykiem USB-C. Teoretyczne przepustowości interfejsów są bardzo wysokie, ale w praktyce o szybkości decyduje kontroler i jakość kości pamięci: wiele tanich pendrive’ów osiąga dobre odczyty, ale bardzo przeciętny zapis.

Typowe wartości spotykane w praktyce to kilkadziesiąt–kilkaset MB/s przy odczycie, natomiast zapis może wahać się od kilku–kilkunastu MB/s w najtańszych modelach do ponad 100 MB/s w lepszych konstrukcjach. Dodatkowo należy pamiętać, że kopiowanie dużego jednego pliku (np. film) jest zwykle szybsze niż kopiowanie wielu małych plików (np. katalogu ze zdjęciami lub projektu programistycznego), ponieważ narzut na operacje plikowe jest wtedy większy.

Warto zwracać uwagę na praktyczne cechy użytkowe: trwałość złącza (zwłaszcza w bardzo małych obudowach), odporność mechaniczna, oraz to, czy producent podaje realne prędkości zapisu. W przypadku przenoszenia danych często lepszym wyborem od pendrive bywa zewnętrzny SSD (na USB), który oferuje wyraźnie większą i stabilniejszą wydajność.

Dyski SSD

Jak już wspomnieliśmy w poprzednim rozdziale, w tym przypadku nazwa jest nieco myląca. Dysk SSD w zasadzie nie jest żadnym dyskiem, w skrócie można powiedzieć, że jest to urządzenie pamięci masowej, w którym rolę nośnika informacji pełnią półprzewodnikowe pamięci flash - składa się wyłącznie z układów pamięci flash oraz elektroniki pomocniczej. Klasyczne dyski twarde to skomplikowane mechanicznie urządzenia, co czyni je bardzo wrażliwymi na urazy mechaniczne. Jeśli dysk twardy spadnie ze stołu, jedyne co będziemy mogli z nim zrobić, to zamieść szczątki do śmieci. W takiej samej sytuacji dysk SSD prawdopodobnie nie ulegnie żadnemu uszkodzeniu. Z tej zalety korzystają przede wszystkim użytkownicy komputerów przenośnych. To głównie tam dyski twarde narażone są na urazy mechaniczne. Brak elementów ruchomych przekłada się nie tylko na wytrzymałość mechaniczną dysków SSD, ale także na ich w zasadzie bezgłośną pracę. No i trzecia zaleta - wysoka sprawność energetyczna. Dysk SSD zazwyczaj pobiera podczas pracy ok 1 W, co jest wartością dziesięć razy niższą niż typowy pobór prądu dysku HDD. Do innych zalet należy brak nagrzewania się, niska waga i potencjalnie małe rozmiary, czy też wysoka odporność na zakłócenia elektromagnetyczne.

Z punktu widzenia bardziej "informatycznego" dyski SSD różnią się diametralnie od rozwiązań mechanicznych w jednym parametrze - czasie dostępu. W praktyce taki dysk jest pamięcią o dostępie swobodnym - stałym dla każdego zapisanego bitu. Nie bez znaczenia jest też wartość czasu dostępu - co najmniej rząd wielkości niższy niż w przypadku klasycznych rozwiązań.

Innym parametrem jest czas odczytu i zapisu danych. I tu ujawniają się pierwsze wady pamięci półprzewodnikowych. W ich przypadku występuje wyraźna asymetria w przypadku obu tych czasów - zapis jest zawsze wielokrotnie wolniejszy niż odczyt, co więcej - odczyt odbywa się szybciej niż w klasycznych dyskach, lecz zapis już niestety jest wolniejszy. Z zapisem na dysk jest związana jeszcze jedna wada urządzeń

półprzewodnikowych - ograniczona liczba cykli (ilości zapisów pojedynczej komórki).

Oba wspomniane parametry (szybkość zapisu i ilość cykli) różnią się znacząco w przypadku dwóch technologii budowania pamięci RAM - MLC (ang. Multi Level Cell) oraz SLC (ang. Single Level Cell). Pamięci typu MLC są tańsze w produkcji oraz mają większą pojemność, natomiast SLC są i droższe, i mają mniejsze upakowanie informacji (mniejszą pojemność), w zamian wytrzymując 10x więcej cykli zapisów, na dodatek ze znacznie większą prędkością.

We współczesnych komputerach spotyka się dwa główne sposoby podłączania SSD. Pierwszy to SSD SATA (złącze jak w dyskach HDD), których wydajność jest ograniczona przepustowością interfejsu – w praktyce sekwencyjny odczyt i zapis zwykle kończą się w okolicach około 500–550 MB/s. Drugi to dyski SSD NVMe korzystające z magistrali PCI Express (najczęściej w formacie M.2). Takie nośniki mogą oferować wielokrotnie wyższe transfery sekwencyjne oraz znacznie lepszą wydajność operacji losowych.

Trzeba też odróżnić format fizyczny od protokołu: M.2 opisuje postać i złącze modułu, ale sam nośnik M.2 może działać zarówno jako SATA, jak i jako NVMe/PCIe. W komputerach osobistych najczęściej spotyka się dziś dyski NVMe pracujące na PCIe x4, a kolejne generacje PCIe (np. 3.0, 4.0, 5.0) zwiększają dostępne pasmo. W praktyce największą różnicę użytkową daje przesiadka z HDD na dowolny SSD; różnice między szybkim SSD NVMe a SSD SATA są zauważalne głównie w zastosowaniach intensywnie dyskowych (kopiowanie dużych plików, obróbka wideo, praca na dużych projektach, wirtualizacja).

Oprócz transferów sekwencyjnych ważnym parametrem jest wydajność operacji losowych (często opisywana jako IOPS), bo to ona wpływa na „responsywność” systemu. Warto też mieć świadomość, że niektóre SSD (zwłaszcza małe lub tanie) mogą zwalniać podczas długotrwałego zapisu, gdy wyczerpie się buforowanie SLC i nośnik zacznie zapisywać dane bezpośrednio w wolniejszym trybie TLC/QLC.

W praktycznych specyfikacjach producenci podają także trwałość TBW (Total Bytes Written). Jest to graniczna ilość danych, jaką można zapisać na nośniku w okresie jego eksploatacji. Dla typowych zastosowań domowych i biurowych wartości TBW w nowoczesnych SSD zwykle okazują się w pełni wystarczające na wiele lat normalnego użytkowania.

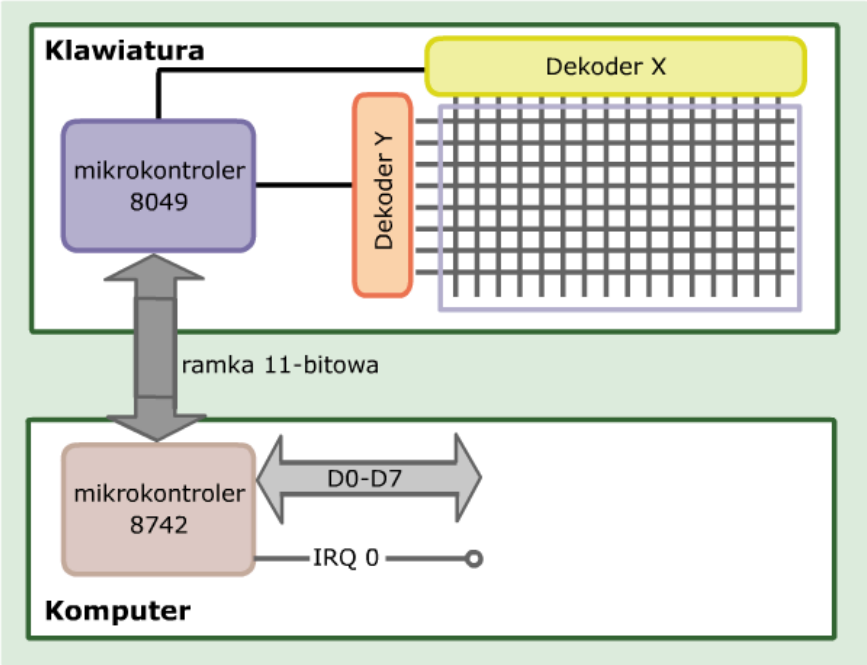
1.5. Urządzenia wejściowe

Klawiatura

Upowszechnienie klawiatury jako urządzenia wejściowego było istotnym krokiem w rozwoju interakcji człowieka z komputerem. Wcześniej wprowadzanie danych odbywało się głównie za pomocą kart perforowanych i terminali znakowych, a sama klawiatura wywodzi się bezpośrednio z konstrukcji maszyn do pisania. Historycznie klawiatury podzielić można na modele XT i AT oraz klawiatury o układzie 101, 102 lub 104 klawiszy.

Klawiatura XT posiadała 83 klawisze i zapewniała jedynie jednokierunkową wymianę informacji pomiędzy klawiaturą a komputerem. Klawiatura AT była wyposażona w 84 klawisze i umożliwiała dwukierunkową komunikację.

Zasada działania klawiatury jest stosunkowo prosta. Klawiatura posiada własny mikrokontroler, który nadzoruje siatkę połączeń linii X i kolumn Y, w węzłach której umieszczone są poszczególne klawisze. Układ wysyła impulsy kolejno wszystkimi liniami X i bada, czy nie pojawiły się one na którejś z linii Y. Dzięki takiemu rozwiązaniu można jednoznacznie określić, które klawisze zostały naciśnięte, a które zwolnione. Informacja o mapie klawiatury przekazywana jest szeregowo do komputera. W interfejsie PS/2 transmisja wykorzystuje osobną linię zegarową, jednak komunikacja ma charakter zdarzeniowy i inicjowana jest przez klawiaturę. Ramka transmisyjna ma długość 11 bitów i zawiera: bit startu, 8 bitów danych, bit kontroli parzystości oraz bit stopu. W klasycznych systemach DOS i BIOS pojawienie się znaku było sygnalizowane przerwaniem IRQ1; we współczesnych systemach operacyjnych obsługa klawiatury realizowana jest przez warstwę sterowników HID. Źródła: IBM PC/AT Technical Reference; Microsoft HID Architecture Overview.



Schemat działania klawiatury

Należy pamiętać, że każdemu klawiszowi przyporządkowany jest unikalny numer (ang. *scan code*), a naciśnięcie klawisza powoduje przesłanie do komputera tzw. kodu naciśnięcia (ang. *make code*). Znaki narodowe nie są generowane sprzętowo przez klawiaturę, lecz wynikają z mapowania kodów klawiszy w systemie operacyjnym i wybranego układu klawiatury.

Układ znaków na klawiaturze jest zależny od przyjętego standardu. W Polsce stosuje się dwa główne układy: układ programisty, w którym polskie znaki uzyskuje się przy użyciu klawisza ALT, oraz układ maszynistki, wywodzący się z tradycyjnych maszyn do pisania (QWERTZ).

Komentarz

Często spotyka się opinię, że układy QWERTY i QWERTZ zostały zaprojektowane w celu utrudnienia szybkiego pisania. W rzeczywistości ich celem było ograniczenie zacinania się dźwigni w mechanicznych maszynach do pisania. Istnieją alternatywne układy, takie jak Dvoraka, zoptymalizowane pod kątem ergonomii, które jednak nie przyjęły się jako standard.

Mysz

Myszkę wynalazł w 1963 roku Douglas Engelbart, pracujący w Stanford Research Institute. W latach 70. technologię tę rozwijała firma Xerox, natomiast popularyzacja myszy nastąpiła wraz z upowszechnieniem komputerów Apple Macintosh i graficznego interfejsu użytkownika.



Mysz firmy Microsoft

Mysz przekazuje do komputera informacje o zmianie swojego położenia, co skutkuje ruchem kursora na ekranie, natomiast naciśnięcie przycisków powoduje wykonanie określonych akcji w systemie lub aplikacji.

Ze względu na sposób odczytu ruchu wyróżnia się myszy mechaniczne, optomechaniczne i optyczne. Modele mechaniczne i optomechaniczne mają dziś znaczenie wyłącznie historyczne. Współczesne myszy wykorzystują sensory optyczne lub laserowe, zdolne do pracy na niemal dowolnym podłożu.

Ważnym parametrem myszy jest jej rozdzielczość, wyrażana w DPI (Dots Per Inch). Współczesne urządzenia oferują wartości od kilkuset do kilkudziesięciu tysięcy DPI, przy czym zbyt wysoka rozdzielczość nie zawsze przekłada się na większy komfort pracy użytkownika.

Większość myszy komunikuje się z komputerem poprzez interfejs USB lub bezprzewodowo, z wykorzystaniem standardu Bluetooth bądź dedykowanych łączy radiowych.

Skaner

Zadaniem skanera jest konwersja dokumentów i zdjęć z postaci papierowej do postaci cyfrowej. Przesyłanie danych do komputera realizowane jest z użyciem standardów takich jak TWAIN, WIA (Windows Image Acquisition) lub SANE w systemach Unix i Linux. Typowym zastosowaniem skanerów jest również rozpoznawanie tekstu (OCR – *Optical Character Recognition*), które we współczesnych rozwiązaniach opiera się na głębokich sieciach neuronowych, w szczególności konwolucyjnych (CNN). Źródła: TWAIN Working Group; Microsoft WIA Documentation; IEEE – Deep Learning for OCR.

Podstawowym parametrem skanera jest rozdzielczość, na którą składają się:

- **rozdzielczość sprzętowa** – zależna od jakości elementów optycznych i czujników; typowe wartości to 600x600 dpi lub 1200x1200 dpi.
- **rozdzielczość interpolowana** – wynikająca z algorytmów przetwarzania obrazu; wartości te nie zawsze wiernie odwzorowują oryginał i mają znaczenie głównie marketingowe.

Informacja o barwie każdego punktu zapisywana jest standardowo na 24 bitach. Profesjonalne skanery stosują większą głębię wewnętrzną (np. 36 bitów), a następnie redukują zakres do formatu wyjściowego. Istotnym parametrem jakościowym jest również zakres gęstości optycznej (D), określający zdolność rozróżniania szczegółów w najjaśniejszych i najciemniejszych partiach obrazu.



Skaner płaski

Konstrukcyjnie skanery dzielą się na urządzenia wykorzystujące układy CCD (Charge Coupled Device) oraz CIS (Contact Image Sensor). Układy CCD oferują wyższą jakość kosztem większych rozmiarów i poboru mocy, natomiast CIS charakteryzują się prostszą budową, mniejszą masą i niższą ceną.

Skanery komunikują się z komputerem głównie poprzez interfejs USB (w tym USB-C) lub sieć komputerową; magistrale SCSI i FireWire mają dziś znaczenie historyczne.

1.6. Urządzenia wyjściowe

Monitor

Współcześnie jako urządzenia wyświetlające obraz z komputera stosuje się głównie monitory LCD i ich pochodne, niemniej niniejszy opis byłby niepełny, jeśli nie uwzględnilibyśmy w nim rozwiązania praktycznie już historycznego - monitorów CRT. Prototypem tych monitorów była, zaprezentowana w 1897 roku, lampa obrazowa typu CRT (ang. *Cathode Ray Tube*), tzw. oscyloskop. Stosowany później kineskop CRT był wciąż lampą elektronową pokrytą od wewnątrz warstwą luminoforu, na którym elektrony rysują obraz z określoną częstotliwością, typowo w zakresie 60-85 razy na sekundę. Wspomnieć należy, że kluczową rolę w rozwoju kineskopów do monitorów odegrał moment upowszechnienia monitorów kolorowych - monitory takie istniały wcześniej, lecz dopiero w latach dziewięćdziesiątych stały się powszechnie dostępne na rynku komputerów osobistych. Źródło: VESA - standardy wyświetlania (przegląd): https://www.vesa.org/
Jedną z własności monitorów jest wielkość plamki (ang. *dot pitch*), czyli odstępu między dwoma sąsiednimi punktami obrazu. Pojęcie plamki wiąże się jednak ściśle z konstrukcją maski kineskopu. Dla masek perforowanych, w których każdy piksel składa się z trzech punktów luminoforu (tzw. triady) w kolorach podstawowych RGB ułożonych w kształt trójkąta, wielkość plamki liczona jest jako odległość między punktami luminoforu tego samego koloru. W przypadku masek szczelinowych i kratowych, o wielkości plamki decyduje szerokość jednej triady luminoforu.



Monitor kineskopowy

Istotnym parametrem monitora są również jego wymiary, w tym głównie przekątna obrazu, która opisuje wielkość przekątnej ekranu (np. 17 cali). W ich konstrukcji uwzględnione były różnice w odległości pomiędzy działem elektronowym a środkiem ekranu, oraz działem a krawędzią kineskopu - kształt plamki na krawędzi zbliżony jest do elipsy, zaś w środku do okręgu. Wyrzutnia elektronowa w połączeniu z układem ogniskującym, dąży do zapewnienia możliwie stałego kołowego przekroju plamki obrazu, niezależnie od miejsca padania wiązki elektronowej na luminofor. Gwarantowało to wysoką ostrość obrazu. Dodatkowo nowe wysokonapięciowe katody (wyrzutnie elektronowe) charakteryzują się mniejszymi wymiarami i zmniejszonym poborem prądu.

Kolejnym parametrem monitorów jest częstotliwość odświeżania (pionowa), która określa, ile razy w ciągu sekundy tworzony jest obraz. Według normy TCO '99 (odnoszącej się do monitorów kineskopowych) ergonomiczne minimum, powyżej którego oko ludzkie nie dostrzega migotania, to 85 Hz. Źródło: TCO Development (standardy historyczne): https://www.tcodevelopment.com/

Konwergencja (zbieżność kolorów) stanowi ważny parametr wyświetlanego obrazu. Konwergencja określa precyzję, z jaką wyświetlane są kolorowe punkty. Każdy piksel tworzony jest z trzech barw składowych (czerwonej, zielonej, niebieskiej). Jeżeli miejsce padania na luminofor którejś z wiązek elektronowych, odpowiedzialnych za rysowanie barw, jest przesunięte względem pozostałych dwóch, to obraz punktu ulega rozmyciu. Parametr ten dotyczy monitorów CRT, a nie monitorów LCD.

Monitor LCD

Głównym składnikiem wyświetlaczy LCD (ang. Liquid Crystal Display) jest pewna substancja, która wykazuje własności zarówno cieczy jak i ciała stałego. Mowa tu o odkrytym w 1888 roku przez austriackiego botanika Friedrich'a Rheinitzer'a ciekłym krysztale.

Kolejna ważna data związana z panelami LCD to rok 1973. To wtedy, za sprawą firmy Sharp, na rynek trafił pierwszy seryjnie produkowany kalkulator zawierający wyświetlacz ciekłokrystaliczny.

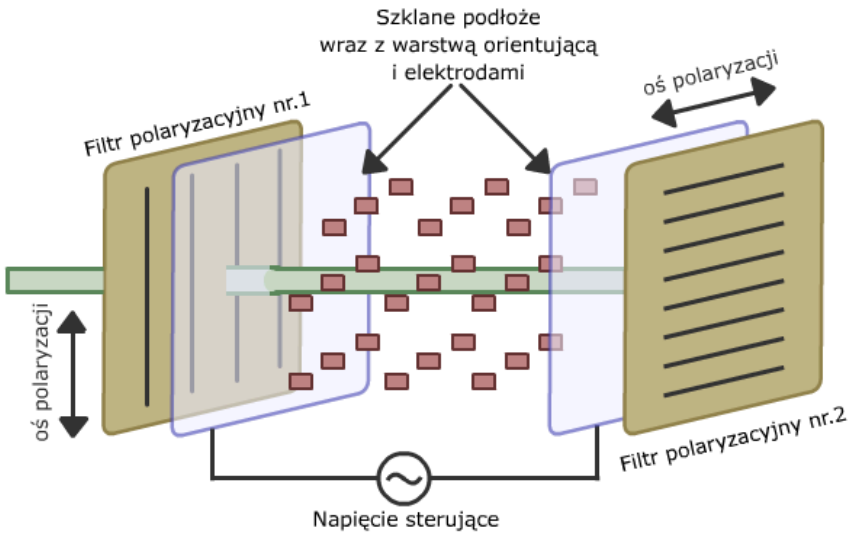
Ogólna zasadę działania paneli LCD opisać można następująco (patrz rysunek poniżej):

1. Źródło podświetlenia emituje światło (w nowoczesnych monitorach LCD podświetlenie realizowane jest niemal wyłącznie z użyciem diod LED, a świetlówki były rozwiązaniem starszym). Źródło: przegląd technologii wyświetlania (Samsung Display): https://www.samsungdisplay.com/

2. Światło przechodzi przez filtr polaryzacyjny nr 1.

3. Spolaryzowane światło dociera do substancji ciekłokrystalicznej, której długie molekuły wcześniej, w procesie produkcyjnym, zostały odpowiednio ułożone wewnątrz każdej z komórek matrycy LCD (o rozmiarze np. 1024 x 768 punktów).
    - W przypadku braku zasilania molekuły ciekłego kryształu układają się w położeniu "skręconym", wymuszonym przez warstwę orientującą znajdującą się wewnątrz każdej z komórek. Powoduje to zmianę polaryzacji światła o 90 stopni.
    - Pod wpływem napięcia sterowania cząsteczki ciekłego kryształu układają się równolegle do linii pola elektrycznego.

4. Światło po przejściu przez warstwę ciekłokrystaliczną pada na filtr polaryzacyjny nr 2, którego oś polaryzacji jest zmieniona o 90 stopni w stosunku do filtru nr 1. Zależnie od braku/występowania napięcia sterującego możliwe są dwa rezultaty działania filtru:
    - Brak napięcia powoduje przepuszczenie światła przez filtr - odpowiada to emisji światła, a więc zapaleniu się piksela.
    - Przyłożenie napięcia sterowania powoduje wytłumieniu światła przez filtr nr 2 - piksel czarny.



Budowa i działanie pojedynczej komórki LCD wykonanej w technologii Twisted Nematic - przypadek występowania napięcia sterującego.

Ad. 2
Polaryzacja światła z użyciem filtru powoduje zorientowanie fali świetlnej tak, aby była zorientowana w jednej płaszczyźnie, którą wyznacza oś filtru.
Ad. 4
Sterowanie wartością napięcia sterującego pozwala na zmianę przestrzennego ułożenia cząsteczek ciekłego kryształu. Dzięki temu możliwe staje się odchylenie płaszczyzny spolaryzowanego światła o kąt różny od 90 stopni (np. 25 stopni), a więc uzyskanie różnych odcieni szarości. Kolorowe wyświetlacze zawierają dodatkową warstwę, w której skład wchodzą barwne filtry RGB (ang. Red Green Blue). Dzięki złożeniu jednego piksel z trzech różnokolorowych komórek matrycy, możliwe staje się wyświetlenie dowolnego koloru.

Główne rodzaje wyświetlaczy LCD:

1. Wyświetlacze pasywne **DSTN** (ang. *Dual Scan Twisted Nematic*). Cechy charakterystyczne to:
    - duża bezwładność - czas potrzebny na ustalenie się wymaganego napięcia sterującego migracją cząsteczek ciekłego kryształu jest duży, co owocuje długim czasem odświeżania obrazu;
    - wzajemne oddziaływanie na siebie ścieżek przewodzących - powoduje to ograniczenie palety barw możliwych do wyświetlenia oraz powstawanie przesunięć obrazu przy dużych kontrastach;
2. Matryca aktywna **TFT** (ang. *Thin Film Transistor*). W roku 1970 po raz pierwszy zastosowano tranzystory wbudowane w ekran ciekłokrystaliczny. Każda komórka matrycy sterowana jest cienkowarstwowym tranzystorem TFT, który reguluje napięcie na elektrodach. Dzięki takiej konstrukcji wyeliminowany zostały niemal zupełnie niekorzystny efekt wzajemnego wpływu ścieżek przewodzących na siebie.
3. Wyświetlacze **IPS** (ang. *In-Plane Switching*). Ten opracowany przez Hitachi w roku 1995 standard stworzył ekran o kącie widzenia przekraczającym 60 stopni. Odmiennie niż przy dwóch poprzednich panelach cząsteczki ciekłego kryształu rozmieszczone są w taki sposób, że przyłożenie napięcia sterowania powoduje wyświetlenie piksela. Wyświetlacze tego typu w przeciwieństwie do matryc TFT (złożonych z oddzielonych komórek) zbudowane są jako jednolite struktury, co zapewnia lepszą jakość obrazu. Wadą tych wyświetlaczy jest stosunkowo długi czas reakcji na zmianę obrazu - duży czas potrzebny do wytworzenia odpowiedniego napięcia sterującego ułożeniem cząsteczek ciekłego kryształu w komórce. (Uwaga: współczesne panele IPS są odmianą aktywnych matryc TFT, oferują kąty widzenia rzędu 178 stopni oraz czasy reakcji porównywalne z innymi technologiami LCD). Źródło: LG Display - IPS (przegląd technologii): https://www.lgdisplay.com/
4. Wyświetlacze **VA** (ang. *Vertical Alignment*). Technologia ta stanowi kompromis pomiędzy matrycami IPS a klasycznymi rozwiązaniami TFT. W panelach VA cząsteczki ciekłego kryształu w stanie spoczynku ustawione są pionowo względem powierzchni ekranu, co powoduje bardzo skuteczne blokowanie światła i umożliwia uzyskanie wysokiego kontrastu oraz głębokiej czerni. Pod wpływem przyłożonego napięcia cząsteczki odchylają się, przepuszczając światło. Matryce VA oferują lepszy kontrast niż IPS, jednak kosztem nieco węższych kątów widzenia oraz potencjalnie dłuższego czasu reakcji w porównaniu do najszybszych paneli IPS. Współczesne

odmiany VA są powszechnie stosowane w monitorach multimedialnych i telewizorach ze względu na korzystny stosunek jakości obrazu do ceny. Źródło: AUO Display Technology Overview: https://www.auo.com/

Wyświetlacze LCD dzięki wielu swym zaletom, do których na pewno należy zaliczyć dużą żywotność (ok. 60 tyś. godzin), niewielką grubość oraz ciągle malejącą cenę, aktualnie praktycznie wyparły już klasyczne monitory CRT (kineskopowe). Ponadto, ponieważ są to urządzenia z samej natury swojego działania cyfrowe (w przeciwieństwie do analogowych monitorów CRT), to wraz z ich upowszechnianiem coraz bardziej popularny staje się nowy standard połączenia monitora z kartą graficzną, nazwany DVI, w którym obraz przesyłany jest cyfrowo, co pozwala na uzyskanie jego wyższej jakości oraz zwalnia nas z korygowania ustawień monitora. (Uwaga: obecnie standardami przesyłu obrazu są przede wszystkim HDMI, DisplayPort oraz USB-C w trybie alternatywnym DisplayPort, natomiast DVI ma dziś znaczenie historyczne). Źródła: VESA DisplayPort: https://www.vesa.org/ ; HDMI: https://www.hdmi.org/

W tym miejscu należałoby jeszcze wspomnieć o monitorach LED - które tak naprawdę nie są nową technologią, od LCD różnią się jedynie sposobem podświetlania ekranu. W przypadku LCD źródłem podświetlenia są (najczęściej dwie) świetlówki, w przypadku monitorów LED - matryca diód LED. To drugie rozwiązanie pozwala na wygaszanie części elementów matrycy w przypadku ciemnych scen, i uzyskiwanie dzięki temu głębokiej czerni, oraz jest bardziej efektywne energetycznie. (Uwaga: w praktyce wygaszane są strefy podświetlenia (local dimming), a nie same piksele matrycy; rozwiązania takie występują m.in. w monitorach mini-LED). Źródło: VESA (przegląd pojęć): https://www.vesa.org/

Nowoczesne technologie wyświetlania: OLED i QLED

Dynamiczny rozwój technologii wyświetlaczy doprowadził do pojawienia się rozwiązań wykraczających poza klasyczne panele LCD. Do najważniejszych z nich należą wyświetlacze OLED oraz QLED, które różnią się zasadą działania i właściwościami użytkowymi.

Wyświetlacze **OLED** (ang. *Organic Light Emitting Diode*) są ekranami emisyjnymi, w których każdy piksel stanowi niezależne źródło światła. Dzięki temu możliwe jest całkowite wygaszanie pojedynczych pikseli, co zapewnia idealną czerń, bardzo wysoki kontrast oraz szerokie kąty widzenia. Brak podświetlenia pozwala również na uzyskanie bardzo cienkich paneli. Do wad tej technologii zalicza się wyższą cenę oraz ryzyko nierównomiernego zużycia pikseli przy długotrwałym wyświetlaniu statycznych obrazów.

Technologia **QLED** (ang. *Quantum Dot Light Emitting Diode*) jest rozwinięciem klasycznych paneli LCD. W tego typu wyświetlaczach stosuje się warstwę kropek kwantowych, które poprawiają odwzorowanie barw i jasność obrazu. QLED nie jest technologią emisyjną – nadal wykorzystuje podświetlenie LED, często w połączeniu z lokalnym wygaszaniem. Rozwiązania te charakteryzują się bardzo wysoką jasnością, długą żywotnością oraz brakiem ryzyka wypalania obrazu. Źródła: OLED Association: https://oled-a.org/ ; Samsung Display (quantum dot): https://www.samsungdisplay.com/



Nowoczesny ultraszeroki monitor QLED

Drukarka

Drukarka, w przeciwieństwie do skanera, zamienia postać dokumentu z cyfrowej na papierową. Drukarki podzielić możemy na trzy grupy w zależności od techniki jaką wykorzystują do drukowania:

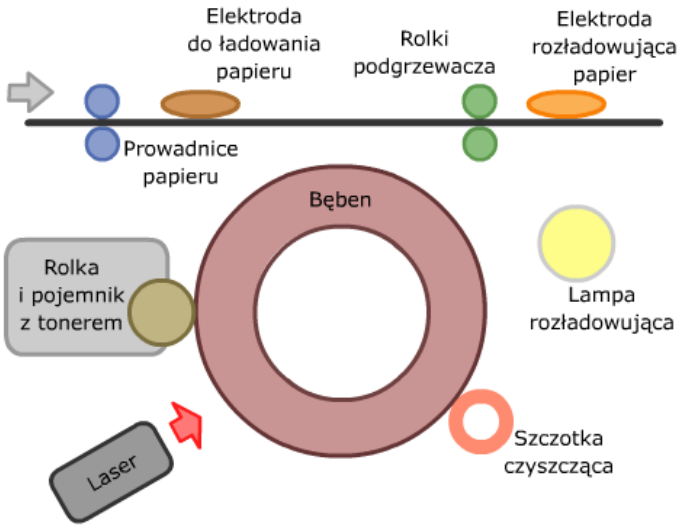
1. **Drukarki igłowe**. Technika druku opiera się na dwóch elementach: głowicy (z igłami) i taśmie barwnej. Igły głowicy uderzając w taśmę barwiącą powodują przeniesienie barwnika na papier. Drukarki tego typu niestety są bardzo głośne i oferują słabą jakość wydruku, i mają w zasadzie jedną zaletę - są w stanie drukować na kalce w więcej niż jednej kopii. Dlatego znajdziecie je jeszcze stosowane jedynie do wydruku faktur, potwierdzeń, itp...

2. **Drukarki atramentowe**. Wykorzystują atrament jako nośnik informacji. Atrament, który znajduje się w zbiornikach, jest doprowadzany do głowicy i wypychany przez dysze. Istnieje wiele rodzajów drukarek atramentowych, wykorzystujących różne techniki do nakładania na papier atramentu. Najpopularniejsze z nich to:
    - technika termiczna - tutaj kropelka atramentu jest "wypluwana" na papier na skutek rozprężenia pęcherzyka powietrza (w każdej z dysz znajduje się specjalny "grzejniczek", który w krótkim czasie potrafi rozgrzać się do wysokiej temperatury. W wyniku takiej reakcji wytwarza się para wodna, której pęcherzyk wypycha kropelkę atramentu z dyszy).



Drukarka atramentowa firmy HP

    - technika piezoelektryczna - w tym wypadku kropelka jest wypychana przez kryształki, które pod wpływem przyłożonego napięcia zmniejszają swą objętość, co powoduje zwiększenie ciśnienia w dyszy i w rezultacie "wyplucie" kropelki atramentu.
3. **Drukarki laserowe**. Zasadę działania drukarki laserowej opiszemy w punktach i zilustrujemy (patrz rysunek 2.16.):
    - Za pomocą specjalnej rolki, toner jest rozkładany w tych miejscach, które wcześniej oświetlił promień lasera - ujemnie naładowany toner jest przyciągany przez dodatnio naładowane punkty bębna.
    - Naelektryzowany toner przyciągany jest przez papier (również naelektryzowany, lecz przeciwnie), po czym zostaje utrwalony przez zespół wałków rozgrzanych do 200 stopni Celsiusza (w temperaturze tej toner topi się).



Zasada działania drukarki laserowej

    - W trzecim etapie bęben jest oczyszczany z resztek barwnika i rozładowywany. Jest gotowy do kolejnego wydruku.

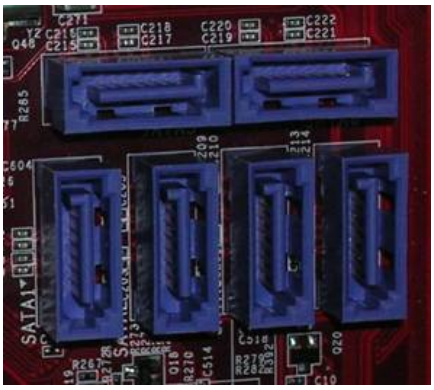
Najważniejszym parametrem drukarki jest maksymalna rozdzielczość, z jaką jest w stanie drukować (rozdzielczość ta może być inna dla wydruku kolorowego, a inna dla czarno-białego). Typowe rozdzielczości popularnych drukarek atramentowych to 600x300 dpi, 600x600 dpi. Oczywiście drukarki laserowe (ale i atramentowe również) oferują znacznie większe rozdzielczości. Drugim, ważnym czynnikiem jest czas drukowania dokumentów (drukarki atramentowe ok. 4-6 stron na minutę, drukarki laserowe 10 i więcej). (Uwaga: współczesne urządzenia często osiągają wyższe wartości, rzędu 10-20 stron/min dla atramentowych i 30-60 stron/min dla laserowych). Źródło: specyfikacje producentów (HP/Brother/Canon): strony produktowe producentów.

Na zakończenie warto wspomnieć, że drukarki komunikują się z komputerem wykorzystując porty USB, lub też korzystając bezpośrednio z interfesów sieciowych (Ethernet, Wi-Fi). Źródło: standardy łączności urządzeń biurowych (przegląd): https://www.ieee.org/

1.7. Interfejsy

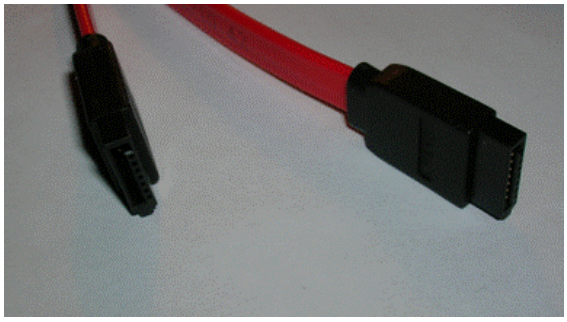
Serial ATA (SATA)

Jest to interfejs do podłączania urządzeń wewnętrznych takich jak dyski twarde HDD, dyski SSD oraz napędy optyczne (np. DVD RAM). Pierwsza wersja standardu SATA została opublikowana w roku 2001. Standard ten wypierał stopniowo standard ATA, który osiągnął swoje maksimum jako medium transmisyjne na poziomie 133MB/s w trybie Ultra DMA. Pierwsza generacja SATA (1,5 Gbit/s) miała przepływność 1,5 Gbit/s na łączu (z narzutem kodowania 8b/10b), co odpowiada teoretycznie 150 MB/s. Źródło: SATA (revisions, przepływności): https://en.wikipedia.org/wiki/SATA

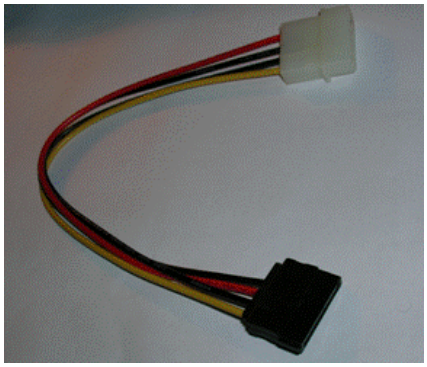


Gniazda kontrolera SATA na płycie głównej

Każde urządzenie SATA podłączane jest osobnym kablem do portu. Standard przewidywał podłączanie „na gorąco” (hot-plug), ale w praktyce wymagało to wsparcia kontrolera i systemu operacyjnego. Kabel sygnałowy zakończony jest wtyczką z siedmioma stykami o różnej długości. Podobnie jak w przypadku USB, w złączach przewidywano rozwiązania minimalizujące ryzyko błędów podczas podłączania. Natomiast złącze zasilające urządzenia SATA ma 15 styków.


Wtyczka złącza Sata

Sata on the Go było rozwiązaniem marketingowym (spotykanym m.in. na płytach głównych), w którym port SATA był wyprowadzany na zewnątrz obudowy komputera w celu podłączania dysków zewnętrznych podczas pracy komputera. W praktyce zewnętrzny wariant SATA upowszechnił się pod nazwą eSATA. Źródło: SATA (warianty i tło historyczne): https://en.wikipedia.org/wiki/SATA


Przejściówka z wtyczki MOLEX do zasilania dysku Sata

W kolejnych wersjach standardu SATA zwiększano przepustowość łącza: SATA 3 Gbit/s (często potocznie nazywany „SATA II”) zapewniał teoretycznie 300 MB/s, natomiast SATA 6 Gbit/s („SATA III”) teoretycznie 600 MB/s. Wersje te wprowadzały także usprawnienia funkcjonalne; do najbardziej znanych należy NCQ (Native Command Queuing) – kolejkowanie poleceń, które mogło poprawiać wydajność szczególnie przy wielu równoczesnych operacjach I/O (np. w systemach wielozadaniowych). Źródło: SATA (revisions, NCQ): https://en.wikipedia.org/wiki/SATA

Warto dodać, że współcześnie najwydajniejsze dyski SSD w komputerach osobistych korzystają zazwyczaj z interfejsu NVMe, działającego po magistrali PCI Express (zamiast SATA). SATA pozostało jednak bardzo powszechne w dyskach HDD oraz w tańszych SSD, a także w systemach, gdzie liczy się kompatybilność i prostota integracji.

PCI Express



gniazdo PCI Express x16. poniżej x1

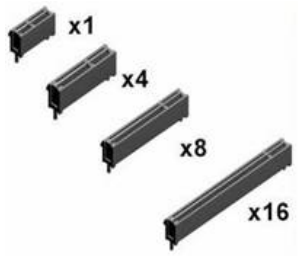
Jest to szeregowa szyna systemowa, służąca głównie do podłączania urządzeń (kart rozszerzeń) do płyty głównej. Standard ten powstał jako następca starszego PCI i AGP i umożliwia przesyłanie danych w obu kierunkach (full duplex – jednocześnie od i do urządzenia). W wersji PCI Express 1.x pojedyncza linia (oznaczana jako x1) oferowała około 250 MB/s na kierunek (teoretycznie), co dawało około 500 MB/s łącznej przepustowości dwukierunkowej. Źródło: PCI Express (tabela porównawcza generacji): https://en.wikipedia.org/wiki/PCI_Express

Zgodnie ze specyfikacją dostępne są połączenia o różnej liczbie linii: x1, x2, x4, x8, x16 i rzadziej x32. W praktyce w komputerach osobistych najczęściej spotyka się gniazda x1 (np. karty sieciowe, kontrolery) oraz x16 (karty graficzne, czasem szybkie karty rozszerzeń). Ważną cechą PCI Express jest kompatybilność wsteczna: karta PCIe zwykle mogła działać w starszym slocie PCIe (z ograniczoną przepustowością), a także w slocie o innej liczbie linii (np. karta x1 w slocie x16), o ile pozwalała na to konstrukcja gniazda i płyty.

W kolejnych generacjach PCI Express znacząco zwiększano parametry łącza. PCIe 2.0 podwoiło szybkość transferu do 5,0 GT/s na linię, co dawało około 500 MB/s na kierunek na jedną linię (x1). PCIe 3.0 zwiększyło transfer do 8,0 GT/s, ale jednocześnie zastosowano wydajniejsze kodowanie 128b/130b (zamiast 8b/10b), co zmniejszyło narzut i pozwoliło uzyskać około 985 MB/s na kierunek na linię. Źródło: PCI Express (encoding i przepustowości): https://en.wikipedia.org/wiki/PCI_Express

Kolejne wersje utrzymały zasadę „mniej więcej podwajania" przepustowości na generację: PCIe 4.0 (16,0 GT/s) to około 1,97 GB/s na kierunek na linię, PCIe 5.0 (32,0 GT/s) to około 3,94 GB/s na kierunek na linię, natomiast PCIe 6.0 wprowadziło istotną zmianę techniczną (m.in. inny sposób transmisji danych, mechanizmy korekcji błędów) i osiąga około 7,88 GB/s na kierunek na linię. Dla popularnego złącza x16 oznacza to teoretycznie odpowiednio około 31,5 GB/s (PCIe 4.0), 63 GB/s (PCIe 5.0) oraz 126 GB/s (PCIe 6.0) w jednym kierunku. Źródło: PCI Express (tabela porównawcza): https://en.wikipedia.org/wiki/PCI_Express

W praktyce znaczenie kolejnych generacji PCIe zależało od zastosowania: karty graficzne przez długi czas nie wykorzystywały w pełni przepustowości x16, natomiast bardzo szybkie dyski NVMe (podłączane zwykle jako PCIe x4) korzystały z coraz szybszych wersji magistrali. PCI Express stał się więc „uniwersalnym kręgosłupem" współczesnego komputera – dla grafiki, pamięci masowych NVMe, sieci (np. karty 10/25/40/100 GbE), a nawet dla tunelowania w interfejsach zewnętrznych (np. Thunderbolt/USB4). Źródło ogólne: PCI-SIG (organizacja rozwijająca PCIe): https://pcisig.com/



Porównanie gniazd PCI Express

Wcześniej stosowany standard - AGP - dziś już przeszedł do historii.



Port AGP x8

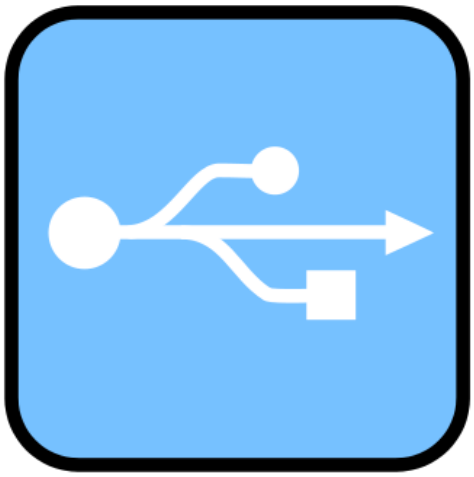
Thunderbolt

W 2009 roku firma Intel we współpracy z Apple zaprezentowała koncepcję interfejsu łączącego cechy PCI Express z DisplayPort (interfejsem używanym do podłączania cyfrowych monitorów). W kolejnych latach standard Thunderbolt rozwijał się generacyjnie: Thunderbolt 3 oraz 4

upowszechniły się przede wszystkim w laptopach i stacjach dokujących, wykorzystując złącze USB-C jako fizyczny konektor i oferując do 40 Gbit/s przepustowości. Thunderbolt był szczególnie przydatny w scenariuszach, w których oprócz przesyłu danych liczyło się także „tunelowanie" PCIe (np. szybkie dyski zewnętrzne NVMe, stacje dokujące z wieloma portami, rozwiązania eGPU) oraz przesył obrazu do monitorów wysokiej rozdzielczości. Źródło: Intel Thunderbolt 4 (opis zastosowań): https://www.intel.com/content/www/us/en/gaming/resources/upgrade-gaming-accessories-thunderbolt-4.html
W nowszej generacji Thunderbolt 5 zwiększono przepustowość do 80 Gbit/s dwukierunkowo, a dodatkowo wprowadzono tryb Bandwidth Boost pozwalający na uzyskanie do 120 Gbit/s w kierunku „do monitora" w scenariuszach wymagających bardzo dużego pasma wideo. Dzięki temu Thunderbolt stał się jeszcze bardziej użyteczny w profesjonalnych zastosowaniach (np. praca z wieloma monitorami, szybkie macierze dyskowe, urządzenia przechwytujące wideo) oraz w nowoczesnych stacjach dokujących. Źródło: Intel – Thunderbolt 5: https://newsroom.intel.com/client-computing/intel-introduces-thunderbolt-5-standard
W praktyce Thunderbolt nie „zastąpił" USB, lecz stał się rozwiązaniem uzupełniającym je w segmencie urządzeń wymagających najwyższej przepustowości i funkcjonalności. Jednocześnie rozwój USB4 sprawił, że część idei znanych z Thunderbolt (np. tunelowanie) zaczęła pojawiać się także w świecie USB, o ile sprzęt i okablowanie spełniają wymagania danej wersji standardu. Źródło: USB4 (ogólny opis): https://en.wikipedia.org/wiki/USB4

USB

Standard USB (ang. *Universal Serial Bus*) to rodzaj sprzętowego portu komunikacyjnego komputerów, zastępującego stare porty szeregowe i porty równoległe. Został opracowany przez firmy Microsoft, Intel, Compaq, IBM i DEC. Na dzień dzisiejszy jest to podstawowy standard podłączania urządzeń peryferyjnych do komputerów osobistych, na tyle uniwersalny, że można go wykorzystać do podłączenia np.: kamery wideo, aparatu fotograficznego, telefonu komórkowego, modemu, skanera, klawiatury, przenośnej pamięci itp.



Logo USB, źródło - Wikipedia.

Standard wspiera łącze typu hot-plug (możliwość włączania / wyłączania urządzenia w trakcie pracy systemu), oraz kilkanaście profili różnych urządzeń. Sama architektura USB jest architekturą klient / serwer, co oznacza, iż zawsze w komunikującej się parze musi istnieć strona nadrzędna (tu nazywana host USB) oraz podrzędna. Host może zarządzać wieloma kontrolerami, a każdy może mieć więcej niż jeden port - co daje możliwość tworzenia mini sieci. Każde urządzenie komunikuje się z hostem korzystając z wydzielonych logicznych kanałów (16 wejściowych, 16 wyjściowych), w sumie teoretycznie host może obsłużyć do 127 urządzeń.

Poprzez złącze USB może także być doprowadzone zasilanie do urządzenia. W starszych wersjach USB typowe wartości były stosunkowo niewielkie (np. 500 mA przy 5V dla USB 2.0), natomiast współcześnie – zwłaszcza przy użyciu złącza USB-C – dzięki USB Power Delivery możliwe jest dostarczanie znacznie większej mocy, nawet do 240W (USB PD 3.1). Źródło: USB-IF – USB Power Delivery (PD 3.1, 240W): https://www.usb.org/usb-charger-pd

USB występuje w kilku podstawowych wersjach:

- USB 1.1 - Urządzenia spełniające warunki tej specyfikacji mogą pracować z szybkością (Full Speed) 12 Mbit/s (1,5 MB/s) i (Low Speed) 1,5 Mbit/s (0,1875 MB/s). Prędkość tego portu nie pozwalała na wygodną pracę z napędami dyskowymi czy kartami sieciowymi, więc w praktyce jego zastosowanie było dość ograniczone.
- USB 2.0 - Urządzenia zgodne z tą specyfikacją mogą pracować z maksymalną szybkością 480 Mbit/s (60 MB/s). Rzeczywista szybkość przesyłu danych zależy od konstrukcji urządzenia, i jest zazwyczaj znacząco niższa, ze względu na wykorzystanie transmisji typu half-duplex na jednej parze przewodów.
- USB 3.0 - Urządzenia zgodne z tą specyfikacją mogą pracować z szybkością 5 Gbit/s (640 MB/s). Rzeczywista przepustowość łącza danych wynosi ok. 4 Gbit/s (narzut kodowania), co w praktyce daje transfery rzędu kilkuset MB/s. Nowy standard oprócz standardowych przewodów (dla kompatybilności w dół z USB 2.0 i 1.1) do szybkich transferów wykorzystuje dodatkowe pary przewodów, co umożliwia pracę w trybie pełnego duplex-u.
- USB 3.1 - Rozwinięcie standardu USB 3.0, w którym (w wariancie 10 Gbit/s) zwiększono maksymalną szybkość transmisji do 10 Gbit/s. W praktyce nazewnictwo USB bywało mylące, ponieważ równolegle funkcjonowały oznaczenia „Gen 1" i „Gen 2", zależnie od przepustowości. Źródło: USB4 / historia i wersje USB (opis ogólny): https://en.wikipedia.org/wiki/USB4

- USB 3.2 - Standard rozwijający rodzinę USB 3.x; w odmianie USB 3.2 Gen 2x2 wykorzystywano dwie „linie” po 10 Gbit/s, co dawało teoretycznie do 20 Gbit/s. Rozwiązanie to spotykało się głównie w szybkich dyskach zewnętrznych i wybranych komputerach. Źródło: Kingston – USB 3.2 Gen 2x2 (opis i przepływności): https://www.kingston.com/en/blog/personal-storage/usb-3-2-gen2x2
- USB4 - Standard oparty o złącze USB-C, umożliwiający 20 Gbit/s i 40 Gbit/s, a w nowszej specyfikacji USB4 Version 2.0 – do 80 Gbit/s. USB4 wprowadza też mechanizmy dynamicznego współdzielenia łącza i „tunelowania” (np. dla danych i obrazu), a wymagania zależą od konkretnej implementacji urządzenia. Źródło: USB-IF – ogłoszenie USB4 Version 2.0 (80 Gbit/s): https://usb.org/sites/default/files/2022-10/USB-IF%20USB%2080Gbps%20Announcement_FINAL_v2.pdf

SLi

nVidia Scalable Link Interface była technologią firmy nVidia umożliwiającą współpracę dwóch kart graficznych w celu uzyskania wzrostu wydajności. Złączenie kart realizowano specjalnym mostkiem wewnątrz komputera. Karty musiały być identyczne, oraz płyta główna musiała wspierać obsługę tego rozwiązania. W praktyce z czasem wsparcie dla SLI malało (zarówno po stronie gier, jak i sterowników), a od 2021 roku zapowiadano wygaszanie profili SLI dla nowych tytułów. Źródło (tło historyczne i ograniczenia): https://www.corsair.com/us/en/explorer/gamer/gaming-pcs/what-happened-to-sli-gpus/

CrossFire

CrossFire było rozwiązaniem firmy ATI/AMD, w którym wzrost wydajności kart graficznych uzyskiwano poprzez połączenie mocy obliczeniowej dwóch kart graficznych. Była to odpowiedź na SLI firmy nVidia. Z czasem rozwiązania multi-GPU stały się niszowe, a nacisk przeniósł się na mechanizmy wieloukładowe wspierane przez API (np. DirectX 12), przy jednoczesnym ograniczaniu „klasycznych” rozwiązań CrossFire. Źródło (tło historyczne, ograniczenie wsparcia w nowszych generacjach): https://www.notebookcheck.net/AMD-confirms-CrossFire-is-no-longer-a-focus-next-gen-Threadripper-to-be-revealed-soon.431301.0.html

IEEE -1394 (FireWire 400)

FireWire był znakiem handlowym firmy Apple - komputery tej firmy standardowo bywały wyposażane w tę magistralę szeregową. Natomiast nazwa IEEE-1394 pochodzi z nr normy amerykańskiego komitetu normalizacyjnego IEEE (The Institute of Electrical and Electronic Engineers).

Ujęte w dokumentach normalizacyjnych cechy magistrali 1394 obejmowały szereg zagadnień jak m.in. prędkości transmisji, topologie i protokoły. Transfer odbywał się w trybie asynchronicznym, ale standard przewidywał możliwość wynegocjowania minimalnej gwarantowanej przepustowości (istotne np. dla wideo).

Zdefiniowane są trzy przepustowości:

- 98.304Mb/s (S100)
- 196.608Mb/s (S200)
- 393.216Mb/s (S400)

Wartości w nawiasach są zaokrągleniami podawanymi często w prasie oraz informacjach na urządzeniach zgodnych z FireWire. I tak na przykład kontrolery PCI pracowały w zakresie S400, kamery zazwyczaj w S100. FireWire było magistralą, co oznaczało że teoretycznie do jednego złącza można było podłączyć aż do 63 urządzeń. W praktyce rzadko spotykało się połączenie więcej niż dwóch urządzeń, tym bardziej, iż cała magistrala pracowała zawsze z prędkością najwolniejszego urządzenia. Źródło: IEEE 1394 (opis ogólny): https://en.wikipedia.org/wiki/IEEE_1394

Fire Wire 800

To nowsza i szybsza wersja interfejsu FireWire znana też jako IEEE-1394b. Jak nazwa wskazuje, można było nią przesyłać dane z prędkością dochodzącą do 800Mb/s (czyli około 100 MB/s w przeliczeniu na bajty, w idealnych warunkach). W stosunku do wcześniejszej wersji standardu liczba urządzeń nie uległa zmianie, podobnie jak topologia połączeń. Wprowadzono natomiast możliwość korzystania ze światłowodów, co mogło wydłużyć dozwoloną długość połączenia. W kolejnych latach FireWire zostało w dużej mierze wyparte przez szybsze wersje USB oraz Thunderbolt. Źródło: IEEE 1394 (tło): https://en.wikipedia.org/wiki/IEEE_1394



Kabel Fire Wire

Wi-fi (Wireless Fidelity)

Skrót ten oznacza technologię tworzenia sieci bezprzewodowych za pomocą fal radiowych o wielkiej częstotliwości. Potocznie nazywane skrótem WLAN (od słów Wireless Local Area Network), z powodzeniem zastąpiły tradycyjne połączenia za pomocą kabli skrętkowych. Specyfikacjami sieci bezprzewodowych zajmuje się komitet IEEE, natomiast Wi-Fi Alliance zajmuje się programami certyfikacji (Wi-Fi 6/6E/7). Współcześnie spotyka się m.in. Wi-Fi 6 (802.11ax), Wi-Fi 6E (rozszerzenie o pasmo 6 GHz) oraz Wi-Fi 7 (802.11be), które zwiększa przepustowość i obniża opóźnienia w porównaniu do wcześniejszych generacji. Źródła: Wi-Fi 6: https://en.wikipedia.org/wiki/Wi-Fi_6 ; Wi-Fi 7: https://en.wikipedia.org/wiki/Wi-Fi_7

Bluetooth

Bluetooth (niebieski ząbek) to standard komunikacji bezprzewodowej, nastawiony nie na tworzenie sieci komputerowej, a na współpracę z urządzeniami peryferyjnymi. W 1994 roku firma Ericsson rozpoczęła prace nad nowym standardem łączenia telefonów komórkowych z innymi urządzeniami. Wspólnie z partnerami założono grupę zainteresowaną rozwojem standardu, i w 1999 roku opublikowano pierwszą jego specyfikację. W założeniach była to sieć bezprzewodowa krótkiego zasięgu o niskim poborze energii. Mechanizmy transportowe w niej zawarte były na tyle uniwersalne by można było przesyłać zarówno pliki jak i dane sterujące oraz strumienie multimedialne (głównie informacji głosowych).

Struktura łącz Bluetooth opiera się o tzw pikosieci - małe, wydzielone sieci składające się maksymalnie z 8 węzłów aktywnych, z których jeden pełni rolę mastera (jednostki nadrzędnej). Dodatkowo w pikosieci może pracować do 255 uśpionych węzłów, pozostających w trybie niskiego poboru mocy i oczekiwania na polecenia od master-a. Wiele pikosieci może równolegle egzystować w tym samym obszarze, bez zakłócania swojej pracy.

BT pracuje na ogólno dostępnycm paśmie 2.400 – 2.4835 GHz. Pasmo to podzielone jest na 79 kanałów. Podczas połączenia kanały zmieniane są okresowo z częstotliwością 1600 skoków na minutę. Skoki te wykonywane są według ściśle ustalonego algorytmu.

Specyfikacja Bluetooth określa trzy klasy nadajników BT:

- Klasa 1 to maksymalna moc promieniowania do 100mW, co daje teoretyczny zasięg w otwartej przestrzeni do 100 metrów.
- Klasa 2 o mocy do 2.5 mW i zasięgu do 10 m była najczęściej wykorzystywana (np standardowo wbudowywana w telefony komórkowe) .
- Klasa 3 to urządzenia promieniujące poniżej 1mW, przy zasięgu do 1 metra.



Adapter USB Bluetooth

Istnieje kilka wersji standardu Bluetooth różniących się głównie prędkością. Najczęściej spotykane rozwiązania to:

- Bluetooth 1.1 o prędkości do 124 kb/s - wystarczającej do strumieniowego przesyłania głosu,
- Bluetooth 2.0 o prędkości do 2,1 Mb/s,
- Bluetooth 3.0 o prędkości 24 Mb/s (wersja 3.1 jeszcze podnosiła prędkość do 40 Mb/s),

W kolejnych latach ważnym krokiem rozwoju stał się Bluetooth Low Energy (LE), wprowadzony w Bluetooth 4.0, który umożliwił budowę urządzeń o bardzo niskim poborze energii (czujniki, beacony, elektronika ubieralna). W praktyce możliwa stała się praca urządzeń zgodnych z Bluetooth LE przez wiele miesięcy na małej baterii, co otworzyło nowe obszary zastosowań. Współcześnie standard Bluetooth jest nadal rozwijany (np. Bluetooth 5.x), a dokumentacja opisuje kolejne usprawnienia techniczne w ramach wersji takich jak 5.4. Źródło: Bluetooth SIG – Bluetooth Core 5.4 (overview): https://www.bluetooth.com/bluetooth-resources/bluetooth-5-4-technical-overview/
Przykładem zastosowań Bluetooth LE były beacony, które były aktywnie promowane na rynku w latach 2010+ (np. firma Estimote). W wielu scenariuszach beacony wykorzystywano do identyfikacji obecności użytkownika w pobliżu urządzenia lub miejsca oraz do prostych interakcji kontekstowych.

Spis treści

1. Zasady doboru konfiguracji sprzętowych
1.1. Definicja wymagań dla systemu komputerowego
1.2. Przykładowe oszacowanie żądanej wydajności systemu serwerowego

1. Zasady doboru konfiguracji sprzętowych

Kolejne etapy wyboru odpowiedniego systemu komputerowego

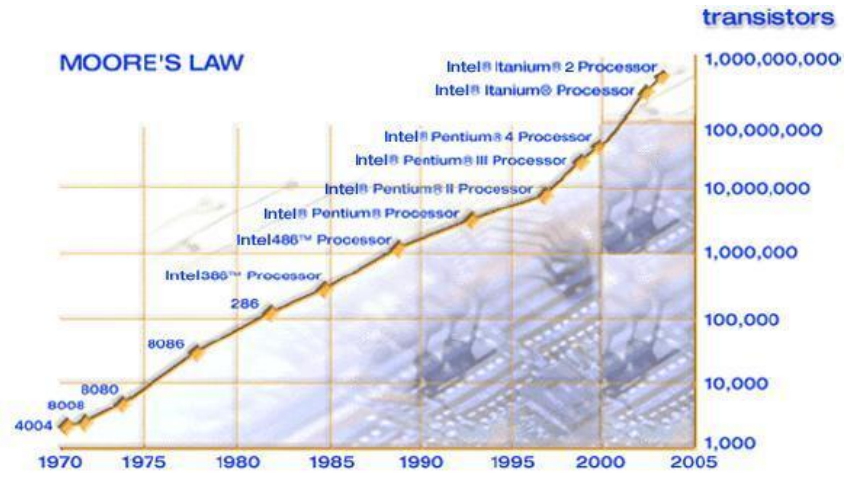
Zanim przystąpimy do analizy istotnych parametrów każdego elementu składowego systemu komputerowego, najpierw chcielibyśmy zaprezentować modelowy proces wyboru sprzętu komputerowego.

System komputerowy zazwyczaj służy do tego, by realizować określone zadania - inaczej będzie kolejnym drogim meblem w pomieszczeniu, ew. grzałką w zimne dni ;). Przez długi czas (szczególnie w naszym kraju) nie zwracano uwagi na fakt, iż zastosowanie systemów komputerowych w jakiejś dziedzinie nie jest wyznacznikiem nowoczesności, wymogiem czasów czy też "prawdą ekranu", lecz ma przynosić konkretne, wymierne korzyści. Ma skrócić czas wykonywania odpowiednich zadań, ograniczyć koszt pracy ludzkiej, zwiększyć bezpieczeństwo i dostępność danych, podnieść jakość produktu itp.

Jednocześnie, zastosowanie systemów komputerowych zazwyczaj pociąga za sobą dodatkowe koszty czy też utrudnienia. W pewnych scenariuszach wprowadzenie nowego systemu (np. w okienkach bankowych lub pocztowych) mogło początkowo zwiększać czas obsługi pojedynczego petenta, zanim personel przyzwyczaił się do procesu i zanim rozwiązania zostały dopracowane. Podobnie, napisanie dokumentu w edytorze tekstu może być wolniejsze niż zapis ręczny, jeśli ktoś nie ma wprawy, natomiast przy poprawkach i archiwizacji często daje przewagę. Zakup sprzętu komputerowego wiąże się również z zakupem oprogramowania i ponoszeniem kosztów jego pielęgnacji (administracji, uaktualnień, modyfikacji), a także z kosztami kopii zapasowych oraz bezpieczeństwa.

Zatem, zamiast podejmować pochopnie decyzję o modernizacji, zastanówmy się czy nam się to opłaca. To jest zadanie bardziej dla menegera niż informatyka, niemniej jednak każdy powinien sobie zdawać sprawę, że komputer nie jest złotym środkiem, który sam z siebie sprawi że praca stanie się łatwiejsza, przyjemniejsza, a życie nabierze odcienia różowego.

Zakup sprzętu komputerowego jest inwestycją - taką jak każda inna, dodatkowo w dziedzinie o szybkim tempie rozwoju technologicznego. W latach 60. Gordon Moore sformułował obserwację dotyczącą tempa wzrostu złożoności układów scalonych, którą po latach zaczęto nazywać "Prawem Moore'a". Warto jednak pamiętać, że jest to obserwacja trendu (a nie prawo fizyki) i bywa upraszczana w postaci popularnych haseł o "podwajaniu mocy". Moore w oryginalnym tekście prognozował szybki wzrost liczby tranzystorów, a później (w 1975) zrewidował tempo do podwajania co około dwa lata. Źródło: Intel - opis i historia Prawa Moore'a: https://newsroom.intel.com/press-kit/moores-law



Prawo Moora w odniesieniu do ilości tranzystorów w procesorze na przestrzeni ostatnich lat

Szybki rozwój sprzętu ma jeden poważny minus - łatwo jest kupić komputer "na wyrost", który nie zwróci się w praktyce w przewidywanym czasie użytkowania. Jednocześnie nie jest prawdą, że każdy komputer "musi" stać się przestarzały po kilkunastu miesiącach: cykl życia zależy od zastosowania, kosztów wsparcia, awaryjności, kosztów energii, licencji, czasu administracji oraz wymagań bezpieczeństwa. W firmach często spotyka się cykle odświeżania rzędu 3-5 lat, ponieważ po latach 4-5 rosną koszty napraw i wsparcia (zwłaszcza poza gwarancją), co może przeważyć korzyści z dłuższej amortyzacji. Źródło (TCO i wpływ kosztów wsparcia w latach 4-5, optymalność cyklu 3-letniego w modelu TCO): Dell - Using TCO to Determine Optimal PC Lifecycle: https://i.dell.com/sites/content/business/smb/en/documents/using-tco.pdf

Pamiętajmy o tym przystępując do zakupów sprzętu komputerowego. Komputery mogą mieć krótki czas efektywnego życia w roli "narzędzia pracy", jeśli wymagania programów i bezpieczeństwa rosną szybko - ale w wielu zastosowaniach mogą pracować dłużej, o ile są utrzymywane, mają wystarczające zasoby i pozostają bezpieczne. Ważny jest też cykl wsparcia systemu operacyjnego: sprzęt może być sprawny, ale jeśli system nie otrzymuje poprawek bezpieczeństwa, rośnie ryzyko utraty danych i infekcji. Dla przykładu wsparcie Windows 10 zakończyło się 14 października 2025 r. i dalsze użytkowanie bez aktualizacji bezpieczeństwa (lub bez dodatkowych programów rozszerzonego wsparcia) zwiększa ryzyko. Źródło: Microsoft - koniec wsparcia Windows 10 (14.10.2025): https://support.microsoft.com/en-us/windows/windows-10-support-has-ended-on-october-14-2025-2ca8b313-1946-43d3-b55c-2b95b107f281

Przebieg procesu doboru sprzętu przedstawimy na prostym przykładzie odnoszącym się do zastosowań z rynku SOHO. Pierwszym krokiem przy wyborze sprzętu komputerowego powinno być jasne określenie jakie funkcje ma on spełniać, jakie zadania mamy zamiar przed nim postawić. Następnie powinniśmy oszacować wymaganą wydajność oraz wielkość zasobów (RAM, dysk) systemu komputerowego, ale też uwzględnić charakter pracy: czy będzie to praca biurowa, nauka, programowanie, grafika, montaż wideo, gry, praca z danymi, wirtualizacja, czy może praca hybrydowa (laptop + stacja dokująca + monitory). Potem zestawiamy przykładową konfigurację, zwracając uwagę na ew. dodatkowe elementy komputera nie związane wyłącznie z wydajnością (monitor, drukarka, dodatkowy osprzęt, ergonomia stanowiska) - nie zapominając także o koszcie niezbędnego oprogramowania, licencji oraz kopii zapasowych. Następnym krokiem powinno być oszacowanie ceny takiego zestawu. Jeśli cena jest zbliżona do dostępnego budżetu - to super. Jeśli jest od niego znacznie niższa - może warto kupić system nieco wydajniejszy (np. więcej RAM i lepszy dysk). Jeśli jest zbyt wysoka - musimy szukać oszczędności, w pierwszej kolejności w elementach peryferyjnych bądź wartościach dodanych, starając się nie psuć kluczowych parametrów (np. nie schodzić zbyt mocno z RAM i jakości dysku, jeśli komputer ma służyć do pracy).

Definicja wymaganej funkcjonalności

Zadania stawiane przed systemem komputerowym można definiować na różnych poziomach dokładności. Optymalną sytuacją jest taka, w której jawnie możemy powiedzieć że potrzebujemy komputera tylko i wyłącznie do jednego celu. Rozważmy taką sytuację:

Jestem programistą i pracuję na kilku typach projektów: aplikacje desktopowe i usługi sieciowe. Do pracy używam środowisk typu IDE (np. Visual Studio / JetBrains), narzędzi do kontroli wersji (Git), a także kontenerów (Docker) oraz czasem maszyn wirtualnych. Część projektów kompiluje się lokalnie, a część w CI/CD (np. na serwerze firmowym lub w chmurze). Poza programowaniem używam komunikatorów, wideokonferencji, przeglądarki z wieloma kartami oraz narzędzi do dokumentacji. Zależy mi na cichej pracy, niezawodności i szybkim dysku SSD, bo kompilacje, indeksowanie kodu i uruchamianie kontenerów mocno obciążają pamięć masową. Pracuję codziennie wiele godzin i chcę mieć dwa monitory. System operacyjny ma być wspierany i bezpieczny; rozważam Windows 11 (często także z WSL2) lub Linux, w zależności od projektów i narzędzi.

Oszacowanie żądanej wydajności i wielkości zasobów

Dzięki tak dokładnemu zdefiniowaniu zadań dla komputera, jesteśmy w stanie szybko i bezproblemowo wybrać komputer. Współcześnie często nie wystarczy sprawdzenie jednego programu, bo typowy "stos" narzędzi to: IDE + przeglądarka (często wiele kart i aplikacje webowe) + komunikatory + narzędzia do kontenerów + testy lokalne + narzędzia graficzne i dokumentacyjne. Dlatego nie bierzemy "minimalnych wymagań", tylko raczej wymagania zalecane (recommended) i dodajemy zapas, bo komputer ma służyć pewien czas, a narzędzia i projekty zwykle rosną.

W praktyce bardzo ważne są: pamięć RAM (aby uniknąć swapowania przy wielu procesach), szybki dysk SSD (najlepiej NVMe) oraz CPU z sensowną liczbą rdzeni (kompilacje, testy i kontenery często wykorzystują wielowątkowość). Jeśli praca obejmuje zadania wymagające GPU (np. obróbka wideo, 3D, część obciążeń związanych z AI), wtedy dobieramy także odpowiedni układ graficzny. Dodatkowo trzeba pamiętać o wymaganiach systemu operacyjnego: przykładowo Windows 11 ma minima (m.in. 4 GB RAM i 64 GB pamięci masowej, UEFI i Secure Boot), ale do pracy programistycznej takie minima zwykle są zbyt niskie i prowadzą do spadków komfortu pracy. Źródło: Microsoft - Windows 11 specs and requirements: https://www.microsoft.com/en-us/windows/windows-11-specifications

Jeżeli chcemy oprzeć się na konkretnym, wiarygodnym przykładzie, możemy sprawdzić wymagania wybranego środowiska programistycznego. Dla RAD Studio dokumentacja instalacyjna podaje m.in. co najmniej 3 GB RAM (8 GB zalecane), procesor dual-core 1.8 GHz (quad-core zalecany), SSD jako mocno rekomendowany oraz kartę zgodną z DirectX 11. To pokazuje skalę "współczesnych" wymagań narzędzi, ale pamiętajmy: finalnie i tak sumują się obciążenia wielu programów uruchomionych jednocześnie. Źródło: Embarcadero - Installation Notes (RAD Studio): https://docwiki.embarcadero.com/RADStudio/Sydney/en/Installation_Notes

Wstępna konfiguracja

Decydujemy się więc na konfigurację zalecaną, czyli:

- komputer ma być przenośny (laptop), ponieważ zakładamy pracę częściowo z domu, a częściowo z firmy; w biurze/lub w domu laptop będzie pracował podłączony do stacji dokującej oraz dwóch monitorów;
- względy prestiżowe czy też warunki przedłużonej gwarancji nie są zbyt istotne - więc będzie to sprzęt "użytkowy" w rozsądnej cenie, ale z porządną gwarancją (bo laptop jako sprzęt mobilny jest bardziej narażony na uszkodzenia niż komputer stacjonarny);
- procesor z serii profesjonalnej (współcześnie: Intel Core Ultra lub Intel Core i5/i7 albo AMD Ryzen 5/7), co najmniej 8 rdzeni logicznych w sensie praktycznym (np. konfiguracje 6–8 rdzeni „dużych” + rdzenie energooszczędne u Intela lub 8 rdzeni u AMD); uzasadnienie: kompilacja, testy oraz narzędzia developerskie i kontenery wykorzystują wielowątkowość;
- płyta główna (w laptopie: platforma) dobrana do procesora, bez specjalnych wymagań odnośnie udostępnianych interfejsów, ale z obowiązkowym: Wi-Fi, Ethernet (bezpośrednio albo przez dock), oraz co najmniej jednym portem USB-C/Thunderbolt/USB4 do stacji dokującej (żeby jednym kablem podłączyć monitory, sieć i peryferia);
- ilość pamięci RAM - 32 GB, ze względu na współczesne narzędzia i pracę wielozadaniową; uzasadnienie (przykładowy „rachunek”): IDE typu Visual Studio ma w dokumentacji zalecenie 16 GB RAM dla typowych profesjonalnych rozwiązań (źródło: Microsoft - Visual Studio 2022 system requirements: https://learn.microsoft.com/en-us/visualstudio/releases/2022/system-requirements), do tego przeglądarka + komunikatory + narzędzia w tle potrafią realnie zająć kolejne kilka GB, a jeśli uruchamiamy 2 kontenery (np. baza danych + backend) to zwykle rezerwujemy im po kilka GB (zależnie od projektu); aby uniknąć swapowania i spadków wydajności przy równoległej pracy (IDE + przeglądarka + kontenery) przyjmujemy 32 GB jako „bezpieczny standard” dla programisty;

- dysk twardy: SSD NVMe 1 TB (system + narzędzia + projekty), ponieważ typowe instalacje środowisk i SDK potrafią zajmować dziesiątki GB, a samo Visual Studio może wymagać (zależnie od zainstalowanych komponentów) nawet bardzo dużej ilości miejsca; Microsoft podaje, że typowe instalacje Visual Studio wymagają około 20–50 GB wolnego miejsca i zaleca instalację na SSD (źródło: Microsoft - Visual Studio 2022 system requirements: https://learn.microsoft.com/en-us/visualstudio/releases/2022/system-requirements);
- napęd DVD-RW: jako element dodatkowy i opcjonalny (zewnętrzny na USB), ponieważ obecnie nośniki instalacyjne i dystrybucja oprogramowania zwykle odbywają się przez sieć i pamięci USB; jeśli jednak ktoś nadal przygotowuje fizyczne nośniki, to zewnętrzny napęd USB rozwiązuje temat bez „pakowania" go do laptopa;
- karta graficzna: zintegrowana (iGPU) jest wystarczająca do pracy programistycznej i obsługi dwóch monitorów, o ile laptop/dock udostępnia odpowiednie wyjścia; dedykowana karta graficzna nie jest tu wymagana, bo nie zakładamy gier/3D jako głównego zastosowania (wyjątkiem byłyby projekty GPU/AI lub praca z wideo);
- dwa monitory LCD do pracy dwumonitorowej: 2 sztuki o przekątnej 27 cali i rozdzielczości co najmniej 2560x1440 (QHD), ponieważ programiści często pracują równolegle z IDE, dokumentacją i podglądem aplikacji; dwa monitory zwiększają produktywność i wygodę pracy (jeden ekran na kod, drugi na dokumentację/uruchomioną aplikację/komunikację);
- stacja dokująca (lub monitor z USB-C jako „dock"): jeden kabel do laptopa ma dostarczać obraz na 2 monitory, sieć i USB dla klawiatury/myszy; to upraszcza pracę „tu i tam" (w firmie i w domu);
- obudowa, mysz, klawiatura - w zasadzie dowolne, byle trwałe; przy pracy wielogodzinnej warto wybrać klawiaturę i mysz wygodne (ergonomia jest praktycznie ważniejsza niż „marka");
- brak dodatkowych, specjalizowanych kart oraz urządzeń wejściowo / wyjściowych;
- system operacyjny + oprogramowanie zabezpieczające + kopie zapasowe; system powinien być wspierany poprawkami bezpieczeństwa (dla przykładu wsparcie Windows 10 zakończyło się 14 października 2025 r., źródło: Microsoft - komunikat o zakończeniu wsparcia Windows 10: https://support.microsoft.com/en-us/windows/windows-10-support-has-ended-on-october-14-2025-2ca8b313-1946-43d3-b55c-2b95b107f281).

Zakup

Mając tak przygotowaną listę możemy przystąpić do skalkulowania ceny w sklepie internetowym bądź w dowolnej firmie zajmującej się sprzedażą komputerów. Jeśli cena jest odpowiednia - dokonujemy zakupu i problem mamy z głowy 😊

1.1. Definicja wymagań dla systemu komputerowego

Tak jasne i klarowne określenie wymagań dla systemu komputerowego, jak to które zaprezentowaliśmy w poprzednim segmencie, niestety rzadko zdarza się w praktyce. Zazwyczaj komputer nie jest wykorzystywany tylko i wyłącznie do jednego celu, lecz chcielibyśmy by był urządzeniem w pełni uniwersalnym. Niestety - uniwersalność kosztuje. Jeśli mamy nieograniczony budżet to wszystko jest ok. Zazwyczaj jednak chcielibyśmy optymalizować koszty, czyli uzyskać sprzęt o wymaganej przez nas funkcjonalności za jak najniższą cenę.

Musimy więc zdefiniować sobie jasno, czego od zestawu komputerowego oczekujemy. Lecz każdy z nas jest inny, każdy może potrzebować czegoś innego i nie istnieje jeden, uniwersalny zestaw cech, które moglibyśmy przejrzeć, przy niektórych z nich postawić znak plus, przy innych minus - i mieć gotowy przepis na zestaw komputerowy. Spróbujemy jednak choć zarysować podział zastosowań systemów komputerowych wraz z podstawowymi wymaganiami w stosunku do każdego z nich. Podstawowy podział klasycznych systemów komputerowych (pomijamy tutaj systemy osadzone oraz superkomputery) z punktu widzenia zastosowania przebiega pomiędzy zastosowaniami infrastrukturalnymi a osobistymi.

Systemy komputerowe w zastosowaniach infrastrukturalnych

Takie zastosowania systemów komputerowych, przy których w normalnym trybie pracy nie siedzi żaden użytkownik, będziemy tutaj nazywali infrastrukturalnymi. Typowym przykładem takiego zastosowania są wszelkiego rodzaju serwery pracujące 24 h na dobę, dostępne jedynie zdalnie i realizujące jakąś usługę dla większej liczby użytkowników, jak np. dostarczanie poczty, stron www, aktualnego czasu, uwierzytelnianie użytkowników sieciowych, itp. Do tego grona zaliczymy również komputery w centrach danych, główne komputery przemysłowych systemów sterowania, systemy monitoringu i archiwizacji danych czy nawet komputery autonomicznych stacji meteorologicznych. Współcześnie bardzo często system infrastrukturalny nie jest pojedynczym komputerem, lecz zbiorem usług uruchamianych na maszynach wirtualnych, w kontenerach lub w środowiskach chmurowych, gdzie kluczowe stają się skalowanie i automatyzacja odtwarzania usług po awarii.

W przypadku takich systemów możemy wyróżnić kilka charaterystycznych wymagań stawianych przed nimi.

Wymagania niezawodnościowe

- Zazwyczaj głównym (a przynajmniej jednym z najważniejszych) wymaganiem jest *niezawodność*. Stosuje się różne miary niezawodności sprzętu komputerowego, najpopularniejszą z nich jest MTBF (ang. *Mean Time Between Failure*), wyrażający średni (statystyczny) czas pomiędzy awariami danego urządzenia. MTBF jest określany na podstawie danych statystycznych lub modeli niezawodnościowych i zwykle podawany w godzinach. Należy pamiętać, że MTBF jest wartością średnią i nie oznacza "czasu, po którym mamy 50% szans awarii" (to częste uproszczenie); przy typowym modelu wykładniczym czas równy MTBF odpowiada sytuacji, w której ok. 63% populacji mogło już ulec awarii. Źródła: Reliability Academy - MTBF a rozkład wykładniczy: https://reliabilityacademy.com/articles/defect-elimination/mtbf-mean-time-between-failure oraz Wikipedia - Mean time between failures (uwagi o błędnych interpretacjach): https://en.wikipedia.org/wiki/Mean_time_between_failures
- W przypadku całego systemu komputerowego niezawodność nie zależy wyłącznie od "najgorszego elementu", lecz od architektury: tego, czy elementy są połączone szeregowo (awaria jednego zatrzymuje usługę), czy też występuje redundancja (np. klaster, replikacja danych, nadmiarowe zasilacze, macierze RAID, load balancer). W systemach infrastrukturalnych analizuje się więc nie tylko MTBF komponentów, ale także czas naprawy, automatyzację odtwarzania usług oraz koszty przestojów.
- Związana z niezawodnością jest również możliwość stosowania *redundancji*, czyli równoległej pracy dwóch lub więcej elementów (serwerów/usług/instancji) w taki sposób, aby awaria jednego nie powodowała przerwania usługi. Redundancja może dotyczyć całych systemów komputerowych (np. układ aktywny-aktywny lub aktywny-pasywny), jak i ich poszczególnych podzespołów (np. zasilacz, pamięć masowa, łącza sieciowe) oraz danych (replikacja, kopie migawkowe).
- Następną cechą, która powiązana jest z dwoma poprzednimi, jest tzw. *hot swap*, czyli możliwość wykonania naprawy systemu komputerowego (wymiany jego elementów) bez zatrzymywania całego systemu, w trakcie jego pracy. W praktyce hot swap dotyczy często dysków w macierzy, zasilaczy, wentylatorów, a czasem także elementów sieciowych w urządzeniach klasy serwerowej.

Powyższe wymagania (a raczej poziom ich spełnienia) definiują nam finalnie stopień osiągalności systemu, czyli procent czasu, w którym realizuje on przedstawione przed nim zadania. W praktyce często spotyka się zapisy w umowach i wymaganiach biznesowych typu "SLA 99.9%" lub "99.99%". Warto pamiętać, że różnice w tych liczbach przekładają się na realny czas przestoju: przykładowo 99.9% oznacza rząd kilku godzin niedostępności w skali roku, a 99.99% to już rząd kilkudziesięciu minut. Źródło: Google SRE - tabela dostępności (downtime vs nines): https://sre.google/sre-book/availability-table/

Przeprowadzając analizę dotyczącą zadań (które muszą być spełnione, a które niekoniecznie), należy zawsze brać pod uwagę koszty ponoszone przez nas w związku z przestojem systemu komputerowego, oraz koszty poszczególnych wymagań. Sam wybór komponentów o lepszych parametrach niezawodnościowych nie gwarantuje ciągłości usługi, jeśli nie zaprojektujemy właściwej architektury (redundancja, kopie, automatyczny failover, monitoring). Zastosowanie pełnej redundancji oraz urządzeń przystosowanych do hot swap jest zdecydowanie droższe - można powiedzieć że jest to inna klasa urządzeń i organizacji pracy. Pamiętajmy o tym zawsze dobierając sprzęt do zastosowań. Redundancja jest np. niezbędna w przypadku nadzorowania procesów rafineryjnych, gdzie każdy przestój instalacji generuje koszt rzędu milionów euro, lecz zazwyczaj chwilowa niedostępność strony internetowej średniej firmy może być akceptowalna, jeśli mamy kopie zapasowe i możliwość szybkiego odtworzenia usługi.

Wymagania wydajnościowe

W następnej kolejności należy postawić wymagania wydajnościowe. Przy czym, analizując wymaganą wydajność sprzętu infrastrukturalnego, zazwyczaj możemy pominąć aspekty związane z wydajnością podsystemu graficznego. Wydajność procesora (procesorów) w systemie infrastrukturalnym także jest analizowana nieco inaczej niż w systemach osobistych. Tutaj nie liczy się zazwyczaj prędkość wykonywania jedego programu, natomiast zależy nam na jak największej wydajności przy pracy z wieloma procesami (programami) oraz na przewidywalnej responsywności usług. Współcześnie w analizie wydajności często równie ważne jak CPU są: ilość pamięci RAM (cache i praca wielu usług), wydajność pamięci masowej (IOPS i opóźnienia), przepustowość i opóźnienia sieci, oraz narzut środowisk uruchomieniowych (wirtualizacja, kontenery). Dodatkowo, tego typu systemy rzadko są kupowane jako systemy generalnego zastosowania, więc możemy oszacować wymaganą wydajność dzięki analizie oprogramowania, które na danym systemie będzie pracowało, oraz przewidywanego obciążenia tego systemu. Dlatego też ważne jest na tym etapie zdecydowanie, jakie oprogramowanie będzie zainstalowane na systemie komputerowym.

Systemy komputerowe w zastosowaniach osobistych

W tym konkretnym przypadku stwierdzenie "zastosowania osobiste" oznacza zarówno zastosowania domowe, jak i firmowe (jako komputer biurowy czy też stacja robocza). Wbrew pozorom, dobór konfiguracji takiego systemu jest trudniejszy niż w przypadku systemów infrastrukturalnych, a przynajmniej trudniejsze jest udowodnienie, iż wybrana konfiguracja spełni pokładane w niej nadzieje. Dzieje się tak, ponieważ od systemów osobistych wymaga się znacznie większej uniwersalności i najczęściej na tym etapie nie jesteśmy w stanie stwierdzić dokładnie, z jakiego oprogramowania będziemy korzystali. Dlatego też proponujemy Wam w tym przypadku podejście uproszczone, czyli odpowiedź na kilka prostych pytań, wybór głównej dziedziny zastosowań systemu komputerowego i w zależności od dokonanego wyboru - powiemy jak oszacować wydajność i jakimi przesłankami się kierować przy dokonywaniu wyboru. Oszacowanie wydajności jest zamieszczone w następnym segmencie tej lekcji.

- Jaki jest przewidywany czas korzystania z systemu komputerowego? Jest to o tyle istotne, iż w przypadku założenia, że ten czas jest dłuższy niż 2-3 lata, musimy liczyć się z tym, że pojawi się nowe oprogramowanie o większych wymaganiach sprzętowych (np. nowsze IDE, przeglądarki, narzędzia do komunikacji). W takim przypadku albo wybieramy system komputerowy z zapasem, albo musimy liczyć się z jego późniejszą rozbudową. Jeśli musimy wybrać system komputerowy z zapasem mocy obliczeniowej, dobrym podejściem jest dołożenie zapasu szczególnie w RAM i pamięci masowej (SSD), bo te elementy najszybciej wpływają na komfort pracy w środowiskach wielozadaniowych.
- Czy przewidujemy rozbudowę komputera? Jeśli tak - to musimy pamiętać, że w komputerach przenośnych rozbudowa bywa ograniczona: w wielu modelach możliwa jest wymiana/rozbudowa dysku SSD, czasem pamięci RAM, ale w cienkich konstrukcjach RAM może być wlutowany i niewymienny. W przypadku komputera stacjonarnego powinniśmy natomiast zwrócić uwagę na dobranie odpowiedniej płyty głównej (zawierającej złącza interfejsów aktualnego standardu, z odpowiednią ilością wolnych gniazd pamięci, itp...). Powinniśmy też dobrać zasilacz z zapasem mocy oraz obudowę, która umożliwi nam włożenie do środka dodatkowych urządzeń.
- Czy chcemy kupić sprzęt stacjonarny czy przenośny? Odpowiedź na to pytanie w oczywisty sposób definiuje nam, kiedy musimy wybrać laptopa (musimy mieć sprzęt przenośny). Lecz w przypadku, gdy decydujemy się na sprzęt wykorzystywany tylko i wyłącznie w jednym miejcu, czasem również warto rozważyć kupno notebook'a. Ten typ komputerów bywa mniejszy i cichszy, ale może być droższy oraz mniej podatny na modernizację niż komputer stacjonarny; z drugiej strony współczesne laptopy potrafią oferować bardzo wysoką wydajność, jeśli dobierzemy je do konkretnych zadań.
- Do jakiej grupy zastosowań będziemy przede wszystkim wykorzystywać system komputerowy? Odpowiedź na to pytanie determinuje nam wymagania sprzętowe dla systemu komputerowego:
    1. zastosowania biurowe - edycja tekstów, obliczenia w arkuszach kalkulacyjnych, przygotowywanie prezentacji, proste bazy danych, itp...;
    2. stacja kliencka sieci komputerowej - korzystanie z internetu (www, ftp, poczta, RSS, komunikatory, itp);
    3. stacja dostępu do usług terminalowych (np. praca zdalna, VDI, Remote Desktop);
    4. wyświetlanie multimediów - odtwarzanie muzyki i filmów;
    5. tworzenie i edycja grafiki;
    6. tworzenie i edycja multimediów (np. montaż wideo);
    7. stacja do gier;
    8. system do nietypowych zastosowań (programowanie, CAD, sztuczna inteligencja, analiza danych, wirtualizacja, itd) - zastosowania nie wymienione wyżej, a wymagające specjalizowanego oprogramowania.
- Czy chcemy korzystać zawsze z najnowszego oprogramowania? Jeśli tak - to otrzymane na podstawie poprzedniego punktu wymagania sprzętowe zwykle warto powiększyć (zapas RAM i SSD ma tu szczególne znaczenie), ponieważ nowe wersje narzędzi i systemów potrafią wymagać więcej zasobów niż wcześniejsze.
- Czy system operacyjny ma dla nas znaczenie? A jeśli tak - to jaki system wybierzemy? Jeśli odpowiedź będzie pozytywna (system operacyjny ma dla nas znaczenie) i chcemy inny system niż Windows, to powinniśmy sprawdzić czy wszystkie komponenty wybrane przez nas będą pracowały pod wybranym systemem (np. dla Linux warto sprawdzić wsparcie sterowników dla Wi-Fi, GPU, czytników linii papilarnych, trybów uśpienia). Praktycznym sposobem jest wybór sprzętu znajdującego się na listach sprzętu certyfikowanego dla danej dystrybucji (np. Ubuntu Certified). Źródło: Canonical - Certified hardware (Ubuntu): https://ubuntu.com/certified
- Czy istotne dla nas są warunki gwarancji oraz obsługi informatycznej przez zewnętrzną firmę? Czy ważne są względy prestiżowe. Jeśli odpowiedź jest pozytywna, to powinniśmy ograniczyć się do sprzętu markowego. W przeszłości na rynku biznesowym silną pozycję miały np. rozwiązania IBM oraz Siemens-Nixdorf, natomiast współcześnie typowe marki biznesowe to np. Dell, HP, Lenovo, Apple i inni duzi producenci. Warto pamiętać, że dział PC firmy IBM został przejęty przez Lenovo, stąd np. komputery ThinkPad są dziś marką Lenovo. Źródło: Lenovo - informacja o zakończeniu przejęcia działu PC IBM (2005): https://news.lenovo.com/pressroom/press-releases/lenovo-completes-acquisition-ibms-personal-computing-division/
- Jaka będzie przeciętna długość pracy przy systemie komputerowym? Odpowiedź na to pytanie jest istotna głównie przy doborze peryferiów. Ergonomia pracy przy komputerze jest jasna - monitor powinien być oddalony o co najmniej pół metra od oczu patrzącego.

Tak więc jeśli będziemy długo pracowali przy komputerze, to nawet w przypadku laptopa warto dokupić zewnętrzny, duży monitor wraz z klawiaturą i myszką, a jeszcze lepiej - stację dokującą lub monitor z USB-C pełniący rolę stacji dokującej.

- Jakich urządzeń peryferyjnych potrzebujemy i jak często będziemy z nich korzystali? Jeśli potrzebujemy drukarki, to zazwyczaj każdy model ma podawane dopuszczalne obciążenie w liczbie stron na miesiąc, a ponadto pamiętajmy, że drukarki laserowe nadal bywają droższe od atramentowych, co może być kompensowane przez niższy koszt wydruku jednej strony. Jeśli będziemy często wykorzystywali skaner, to pamiętajmy, że tanie urządzenia są zazwyczaj mniej wydajne niż te klasy średniej; różnice dotyczą m.in. szybkości skanowania, jakości oraz wygody (np. automatyczny podajnik dokumentów, skanowanie dwustronne), co w praktyce może oznaczać dużą różnicę w czasie pracy.

1.2. Przykładowe oszacowanie żądanej wydajności systemu serwerowego

Przykład takiego oszacowania (za artykułem Piotra Wrzalika zamieszczonym w PC Kurierze).

Naszym celem jest dobranie platformy sprzętowej dla serwisu WWW, o zawartości tworzonej dynamicznie na podstawie bazy danych. Planowanie wydajności lub zdolności obsługi (*capacity*) jest procesem ciągłym i powinno się zacząć w pierwszym etapie realizacji serwisu WWW już wówczas można uzyskać pierwsze prognozy co do liczby wizyt w danym okresie czasu. To z kolei może posłużyć do stworzenia prostego i niedoskonałego jeszcze modelu analitycznego, który przy założeniu kilku empirycznych lub katalogowych wskaźników wstępnie przybliży zapotrzebowanie na wydajność.

Model uproszczony

Dla przykładu przeprowadźmy wstępną analizę dla następujących przewidywanych warunków:

- dziennie będzie składanych ok. 50 000 wizyt, każda potrwa średnio 15 minut;
- serwis będzie przygotowany w języku PHP i w całości dynamiczny;
- użytkownik w czasie pojedynczej wizyty odwiedzi 20 stron.

Resztę parametrów możemy przyjąć domyślnie (a dokładniej - ustalić z projektantem systemu):

- średnia objętość kodu strony HTML - 40 kB (wartość przykładowa);
- średnia objętość grafiki - 200 kB (wartość przykładowa);
- średnia liczba grafik pobieranych z każdą stroną HTML - 6 (część wcześniej pobranych grafik przeglądarki zwykle przechowują w pamięci cache);
- średnia objętość zasobów statycznych innych niż grafiki (CSS/JS/fonty) przypadająca na jedną stronę - 150 kB (wartość przykładowa); (Uwaga: rząd wielkości całkowitego payloadu strony WWW można odnieść do danych z HTTP Archive; mediana payloadu bywa na poziomie ok. 1,7-1,9 MiB.) Źródło: DebugBear (na bazie HTTP Archive): https://www.debugbear.com/blog/avoid-enormous-network-payloads
- stosunek liczby odwiedzin w okresie godzin szczytu (4 godziny - peak load) - 500 proc. obciążenia bazowego;
- średnia liczba skryptów PHP przetwarzana przez jeden rdzeń procesora (lub vCPU w chmurze) - 50 skryptów/sekundę (wartość przykładowa; w praktyce zależy od frameworka, cache, bazy danych i charakteru zapytań);
- średnie wymaganie na pamięć podczas wykonywania skryptu: ok 120 MB / skrypt (wartość przykładowa; w praktyce zależy od aplikacji i konfiguracji PHP, np. limitu pamięci i użycia cache);

Teraz możemy wyliczyć:

- średnia "waga" strony (HTML + grafiki + pozostałe zasoby statyczne): 40 + 6 * 200 + 150 = 1390 kB;
- średnia liczba użytkowników odwiedzających witrynę w tym samym czasie jest możlwa do obliczenia na podstawie całościowego obciążenia. Wg założeń - przez 4 godziny mamy obciążenie 5 razy większe niż bazowe, w pozostałym czasie (20 godzin) bazowe - i to składa się na 50 tyś wizyt. A więc: 50 000 = 20*x + 4*5*x, gdzie x - obciążenie w godzinach normalnej aktywności (bazowe);
- obciążenie normalne x = 1250 odwiedziny/godzinę;
- obciążenie szczytowe 5*x = 6250 odwiedzin/godzinę;
- 1250 * 15 minut / 60 minut = ok 313 użytkowników witryny jednocześnie w godzinach normalnej aktywności;
- 6250 * 15 minut / 60 minut = ok 1563 użytkowników witryny jednocześnie w godzinach szczytowej aktywności;
- średnia liczba równocześnie wykonywanych skryptów PHP:
    - obciążenie normalne: 1250 * 20 / 3600 daje ok. 7 skryptów/sekundę,
    - obciążenie szczytowe: 6250 * 20 / 3600 daje ok. 35 skryptów/sekundę;
- średnia wymagana przepustowość serwera i sieci:
    - w godzinach normalnej aktywności: 1250 * 20 * 1390 kB = 34 750 MB/godz. = ok. 9,65 MB/s,
    - w godzinach szczytowej aktywności: 6250 * 20 * 1390 kB = 173 750 MB/godz. = ok. 48,26 MB/s.

Wszystkie powyższe obliczenia są szacunkowe - więc często zaokrąglane w górę. Ponadto w tej analizie w ogóle nie uwzględniamy czasu potrzebnego na odczyt danych z bazy, czy obsługę sytuacji błędnych oraz zadań administracyjnych (backup, mirroring, itp). Z tego względu, oraz ze względu na nieuniknione pomyłki w obliczeniach przyjmujemy jako docelowe obciążenie o 100 % wyższe niż maksymalne przewidywane w obecnej sytuacji, przy czym limit jest wyznaczany przez wymagania szczytowe.

Podsumowując obliczenia, możemy stwierdzić, że:

- potrzebujemy co najmniej 4 rdzeni procesora / 4 vCPU (35 skryptów / sek przyjmując 50 skryptów/sek na rdzeń daje ok. 1 rdzeń, zwiększone o 100 % daje 2; następnie doliczamy narzuty: baza danych, TLS, logowanie, zadania tła, skoki obciążenia oraz fakt, że w chmurze vCPU nie zawsze zachowuje się jak „pełny rdzeń” - stąd praktyczne minimum 4);
- równoczesne wykonanie 70 skryptów (35 w szczycie powiększone o 100 %) wymaga ok. 70 * 120 MB = 8400 MB RAM. Do tego należy dołożyć wymagania systemu operacyjnego, serwera WWW i zapas bezpieczeństwa (np. 2-4 GB) - czyli potrzebujemy ok. 11-12 GB RAM; w praktyce oznacza to co najmniej 16 GB (ze względu na typowe „progi” konfiguracji i zapas na cache);

- Serwer musi być podłączony za pomocą łącz o przepustowości co najmniej ok. 96,5 MB/s (48,26 MB/s powiększone o 100 %), czyli ok. 772 Mbit/s - w praktyce oznacza to minimum 1 Gbit/s, a dla komfortowego zapasu (i równoległych zadań administracyjnych) warto rozważyć 2,5 Gbit/s lub zastosowanie CDN dla statycznych zasobów.

W niniejszej analizie brak jest oszacowania na przestrzeń dyskową, wymagań odnośnie zasilania, redundancji, itp ... to powinno się znaleźć w Waszych projektach. Bardziej zależało nam na pokazaniu toku myślenia.

Przykład odniesienia do serwerów VPS / chmury

W przypadku VPS lub chmury publicznej powyższe „wyniki" często mapuje się bezpośrednio na parametry instancji: liczba vCPU, ilość RAM oraz przepustowość sieci. Typowy, prosty sposób podejścia jest następujący: wybieramy instancję np. 4 vCPU / 16 GB RAM jako start, a następnie skalujemy pionowo (większa instancja) lub poziomo (więcej instancji) na podstawie metryk (CPU, czas odpowiedzi, liczba żądań na sekundę, opóźnienia bazy danych, cache hit ratio).

Jeśli planujemy skalowanie poziome (np. kilka serwerów aplikacyjnych za load balancerem), kluczowe jest, aby aplikacja była możliwie bezstanowa (stateless) - wtedy dowolna instancja może obsłużyć dowolne żądanie. Dane sesyjne i stan aplikacji przenosi się do usług zewnętrznych (np. baza danych, cache, magazyn obiektów). Źródło: AWS Well-Architected Framework - „Make systems stateless where possible": https://docs.aws.amazon.com/wellarchitected/latest/framework/rel_mitigate_interaction_failure_stateless.html

W praktyce, aby odciążyć serwer aplikacyjny i łącze, często wynosi się statyczne zasoby (grafiki, pliki JS/CSS, fonty) do CDN, który przejmuje dużą część ruchu i redukuje ruch do serwera źródłowego. Źródło: Cloudflare Cache (dokumentacja): https://developers.cloudflare.com/cache/

Model zaawansowany

W modelu uproszczonym brakuje dokładnej analizy wpływu pojedynczej wizyty na serwerze (i jej poszczególnych etapów) na ogólną wydajność systemu oraz określenia szczegółowego kosztu obsługi (*service cost*) związanego z wykorzystaniem procesora, pamięci, dysku i interfejsu sieciowego. Taka analiza jest niezbędna do zbudowania modelu wydajnościowego.

Metodologia zaawansowanego planowania wydajności powinna uwzględniać sześć kroków:

- analizę środowiska pracy serwisu;
- charakterystykę typów obciążenia;
- przygotowanie analitycznego modelu wydajnościowego;
- weryfikację modelu;
- prognozowanie obciążenia;
- zapewnienie odpowiedniej wydajności.

Do stworzenia modelu wydajnościowego niezbędna jest znajomość profilu przeciętnej wizyty użytkownika w naszym serwisie. Najlepszym źródłem tych danych są statystyki działania serwisu, jednak nie zawsze możemy sobie pozwolić na uruchomienie serwisu na małą skalę, by takie statystyki zebrać i na ich podstawie przygotować się do zwiększonego obciążenia. W takiej sytuacji należy przygotować symulację przynajmniej kilkudziesięciu wizyt imitujących normalną pracę użytkownika w serwisie. Do stworzenia opisu profilu przeciętnej wizyty może nam posłużyć dowolny program analizujący logi serwera WWW, np. WebTrends Log Analyser.

Wymagania wydajnościowe dla systemów komputerowych w zastosowaniach osobistych

Tak jak wspomnieliśmy w poprzednim rozdziale, trudno jest oszacować wymagania na moc obliczeniową dla komputerów. W zasadzie nie jest możliwe podanie dokładnych wyliczeń (jak to zrobiliśmy w paragrafie powyżej dla serwerów), jedyne co możemy zrobić, to podać ogólne wytyczne — możliwie konkretne, ale nadal „praktyczne", a nie laboratoryjne.

- Zastosowania biurowe. W tym przypadku wymagania odnośnie mocy obliczeniowej są w dużej mierze definiowane poprzez system operacyjny i typową wielozadaniowość (przeglądarka + pakiet biurowy + komunikator). Dla współczesnych systemów, takich jak Windows 11, minimum formalne jest niskie (m.in. 4 GB RAM), ale w praktyce komfort pracy wymaga wyraźnego zapasu. Sensowna konfiguracja biurowa to: procesor 4–6 rdzeni, 16 GB RAM (żeby przeglądarka z wieloma kartami i Office nie „dusiły" systemu) oraz dysk SSD (najlepiej NVMe) 512 GB. Akcelerator graficzny może być zintegrowany — ważne jedynie, by obsłużył rozdzielczość monitora i ewentualnie dwa wyjścia obrazu. Źródło: Microsoft – Windows 11 specifications: https://www.microsoft.com/en-us/windows/windows-11-specifications
- Stacja kliencka sieci komputerowej — sytuacja w tym wypadku jest podobna do zastosowań biurowych, ale częściej dochodzą wideokonferencje, praca w chmurze i cięższe strony WWW. Dlatego jako „bezpieczne minimum komfortu" przyjmujemy: 16 GB RAM, SSD 512 GB, procesor 4–6 rdzeni oraz kamera/mikrofon o sensownej jakości. Dodatkowo warto mieć stabilne łącze przewodowe (Ethernet) lub Wi-Fi nowej generacji, bo to realnie wpływa na komfort pracy bardziej niż „kolejne 10% mocy CPU".
- Stacja dostępu do usług terminalowych — podobnie jak w dwóch poprzednich przypadkach, ale kluczowa staje się sieć i opóźnienia. Sprzęt lokalny może być umiarkowany (np. 4 rdzenie, 8–16 GB RAM, SSD 256–512 GB), natomiast warto zadbać o: stabilny Ethernet 1 Gbit/s (lub dobre Wi-Fi), dwa monitory (częsty scenariusz pracy z pulpitem zdalnym) oraz wsparcie sprzętowe dekodowania wideo, jeśli używany protokół/środowisko zdalne korzysta z kompresji obrazu.
- Wyświetlanie multimediów. Współcześnie typowym „testem" stało się wideo 4K (a czasem 4K HDR) oraz nowoczesne kodeki, w tym AV1. W praktyce oznacza to, że ważniejsza od „surowego CPU" bywa obecność sprzętowych dekoderów w układzie graficznym (zintegrowanym lub dedykowanym). Sensowna konfiguracja multimedialna to: 4 rdzenie, 8–16 GB RAM, SSD (choćby 256–512 GB) oraz

grafika z obsługą dekodowania nowoczesnych formatów wideo. Źródło (przykład wsparcia sprzętowego, kodeki i akceleracja): NVIDIA – Video Codec SDK: https://developer.nvidia.com/video-codec-sdk

- Tworzenie i edycja grafiki. Tutaj pojawia się pierwsze rzeczywiście wymagające zastosowanie, i to zarówno pod względem szybkości procesora, jak i zapotrzebowania na pamięć operacyjną oraz szybki dysk. W praktyce, dla pracy w narzędziach klasy Adobe Photoshop rozsądne minimum komfortu to 16 GB RAM, ale przy większych plikach i wielu warstwach typową rekomendacją jest 32 GB RAM. Do tego dochodzi szybki SSD NVMe (np. 1 TB, bo pliki robocze i „scratch disk" potrafią urosnąć) oraz GPU z własną pamięcią (praktycznie 6–8 GB VRAM daje komfort przy większych rozdzielczościach i akceleracji). Źródło: Adobe – Photoshop system requirements: https://helpx.adobe.com/photoshop/system-requirements.html
- Tworzenie i edycja multimediów (wideo) jest następnym wymagającym obliczeniowo zadaniem. W porównaniu do grafiki rastrowej, jeszcze większego znaczenia nabiera wydajność wielowątkowa CPU, przepustowość dysku oraz akceleracja kodowania/dekodowania wideo. Sensowna stacja do montażu wideo (zwłaszcza 4K) to: procesor 8 rdzeni (lub więcej), 32 GB RAM (a przy cięższych projektach/8K — 64 GB), dysk NVMe na system i „cache" (1 TB) oraz osobny szybki dysk na materiały. Warto też wybrać GPU z akceleracją kodowania (sprzętowe enkodery), bo to realnie skraca eksport materiału. Źródło: Adobe – Premiere Pro system requirements: https://helpx.adobe.com/premiere-pro/system-requirements.html
- Stacja do gier — tutaj nadal „apetyt rośnie w miarę jedzenia", ale realia są inne niż kiedyś: standardem stały się SSD (często NVMe), 16 GB RAM jako rozsądne minimum oraz mocna karta graficzna (często z 8–12 GB VRAM lub więcej, zależnie od rozdzielczości i ustawień). Sensowna konfiguracja do grania w nowe tytuły to: 6–8 rdzeni CPU, 16–32 GB RAM, SSD NVMe 1 TB oraz GPU z „zapasu mocy" (bo gry szybko go konsumują). Warto też pamiętać o alternatywie: współczesne konsole, takie jak PlayStation 5 czy Xbox Series X, oferują stałą platformę sprzętową i dłuższą „żywotność" konfiguracji względem typowego PC, bo gry są pod tę platformę optymalizowane. Źródło (specyfikacje, ogólny opis platformy): Wired – PS5 specs: https://www.wired.com/story/ps5-specs/, The Verge – Xbox Series X: https://www.theverge.com/2020/3/16/21182703/xbox-series-x-specs-gpu-cpu-ram-storage-tfops-features-4k-ray-tracing
- System do nietypowych zastosowań (programowanie, CAD, sztuczna inteligencja, itd) — mowa o zastosowaniach nie wymienionych wyżej, a wymagających specjalizowanego oprogramowania. Tu sprawę mamy ułatwioną: sprawdzamy wymagania sprzętowe narzędzi i dobieramy sprzęt pod „wąskie gardło" (CPU, RAM, GPU, dysk, sieć). Przykładowo: do lokalnych zadań AI często kluczowa jest karta GPU i zgodność ze stosem obliczeniowym (CUDA po stronie NVIDIA lub ROCm/HIP po stronie AMD). Źródło: NVIDIA – CUDA Toolkit: https://developer.nvidia.com/cuda/toolkit, AMD – ROCm documentation: https://rocm.docs.amd.com/en/docs-7.0.0/index.html

Jak widzicie, każde z zastosowań komputerów osobistych ma swoje specyficzne wymagania i to w zależności od nich powinniśmy dobierać system komputerowy. Komputery PC są rozwiązaniami uniwersalnymi, co wcale nie znaczy że optymalnymi - do każdego zadania optymalna konfiguracja będzie nieco inna. Sporo też można zaoszczędzić analizując swoje potrzeby i dobierając system który je spełnia w miejsce ulegania magii gigaherzów i megabajtów z reklam.

Spis treści

1. Podstawowe informacje o systemie operacyjnym
1.1. Funkcje systemu komputerowego
1.2. Budowa systemu operacyjnego
1.3. Klasyfikacja systemów operacyjnych
1.4. Podsumowanie

1. Podstawowe informacje o systemie operacyjnym

Wstęp

Zakończyliśmy właśnie część podręcznika poświęconą sprzętowi komputerowemu, od tej pory głównym tematem naszych rozważań będzie oprogramowanie. Oprogramowanie podstawowe i kluczowe dla działania systemu komputerowego - czyli system operacyjny.

Wprowadzenie do tej niezwykle bogatej tematyki jaką są systemy operacyjne komputerów zaczniemy od podstawowych informacji dotyczących ich ogólnej charakterystyki, roli i zadań oraz budowy wewnętrznej (to ta lekcja). Potem postaramy się opisać nieco dokładniej kluczowe mechanizmy systemu operacyjnego, a cały blok zakończymy przedstawieniem współczesnych systemów operacyjnych oraz konfiguracją stacji roboczej.

Chcielibyśmy uświadomić Wam rolę systemu operacyjnego w funkcjonowaniu wszelkiego rodzaju systemów komputerowych. Chcielibyśmy także, byście rozumieli, jak są zbudowane najpopularniejsze systemy takie jak Windows czy Linux, byście potrafili sami ocenić, które ze specyficznych rozwiązań w nich zawartych są dla Was istotne, i na koniec - by w trakcie Waszej dalszej edukacji informatycznej podstawowe pojęcia z tej dziedziny - takie jak proces, wątek czy jądro - były dla Was jasne.

Co to jest system operacyjny?

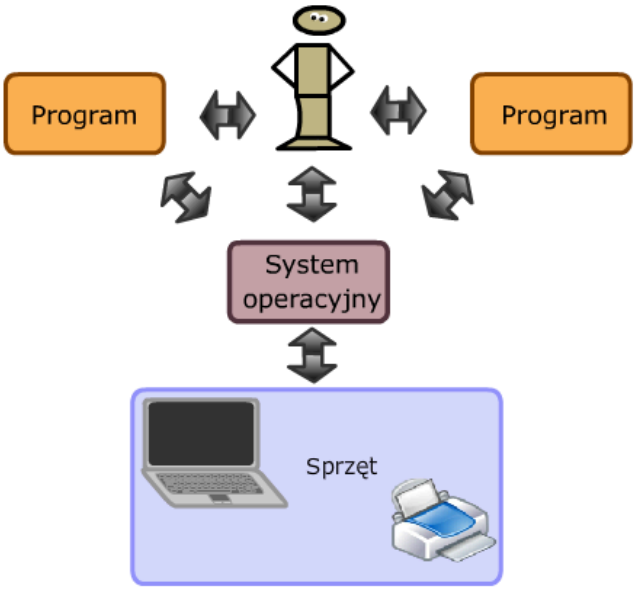
Oto najkrótsza możliwa definicja: system operacyjny (ang. *operating system*) jest **najważniejszym** programem dla naszego komputera ;). A bardziej formalnie, rozpoczniemy od przytoczenia definicji systemu operacyjnego podanej w książce A.Silberschatza, P.Galvina "Podstawy systemów operacyjnych" (WNT 2000):

> ### Definicja
>
> System operacyjny jest programem, który działa jako pośrednik między użytkownikiem komputera a sprzętem komputerowym. Zadaniem systemu operacyjnego jest tworzenie środowiska, w którym użytkownik może wykonywać programy. [...] Możemy uważać system operacyjny za dystrybutora zasobów. System komputerowy ma wiele zasobów (sprzęt i oprogramowanie), które mogą być potrzebne do rozwiązania zadania. [...] System operacyjny pełni funkcję zarządcy owych dóbr i przydziela je poszczególnym programom i użytkownikom wówczas, gdy są one nieodzowne do wykonywania zadań.

Jest to jedna z wielu definicji, które można znaleźć w literaturze. Pojęcie systemu operacyjnego jest trudne do zdefiniowania w zwartej, lakonicznej formie. Najczęściej krótki opis jest zbyt ogólny, żeby uzyskać wyobrażenie o roli i sposobie działania tego specyficznego oprogramowania. Podobnie, trudne jest określenie elementów składowych systemu operacyjnego, czyli jednoznaczne oddzielenie oprogramowania systemowego od aplikacyjnego.

Zazwyczaj najważniejszą cechą systemu operacyjnego (tym, co go odróżnia od innych programów), jest owo "pośredniczenie" między użytkownikiem komputera a sprzętem komputerowym. Jest on niezbędny do prawidłowej pracy komputera i w pewnym sensie określa możliwości wykorzystania sprzętu oraz komfort pracy użytkownika. System operacyjny musi być oczywiście dostosowany do sprzętu, na którym jest instalowany. Od niego zależy również w dużym stopniu jaką pracę możemy wykonać lub jakie programy uruchomić. Wprowadzenie systemu operacyjnego jako warstwy pośredniczącej, oprócz zdjęcia z programistów końcowych niebanalnego problemu oprogramowywania (w ramach każdej aplikacji!) sprzętu, pozwoliło na wprowadzenie pewnej standaryzacji dzięki czemu np. kolejne linie urządzeń są wzajemnie kompatybilne i aplikacje końcowe działające na jednym modelu można najczęściej uruchamiać na jego następcach, bez względu na różnice sprzętowo-architektoniczne. Dobrym przykładem tego ułatwienia jest np. dowolny program do odtwarzania muzyki z plików mp3 działający pod kontrolą systemu Microsoft Windows – programista nie musi przewidywać jaki model karty dźwiękowej znajdzie się w komputerze na którym program ma działać, nie musi też zastanawiać się, jak działa konkretny sterownik ani jakimi szczegółami sprzętowymi posługuje się urządzenie. Chcąc odtwarzać dźwięk, odwołuje się do usług systemu operacyjnego (interfejsów API), a system wraz ze sterownikami przekazuje dane do właściwego urządzenia. (os-book.com)

Dzięki zastosowaniu systemu operacyjnego możliwe jest łatwe wykorzystanie komputerów o różnej architekturze wewnętrznej, zbudowanych z różnych podzespołów różnych producentów, gdyż z punktu widzenia użytkownika obsługa wszystkich podzespołów przeznaczonych do wykonywania jednego zadania jest taka sama. Szczegółami obsługi urządzeń wewnętrznych zajmuje się właśnie system operacyjny. Łatwiej mają również twórcy oprogramowania, którzy nie muszą dostosowywać swoich programów do wielkiej liczby różnorodnych rozwiązań sprzętowych dostępnych na rynku.



System operacyjny jako pośrednik

Do najczęściej spotykanych zadań systemu operacyjnego należą m.in.: zarządzanie procesami i wątkami (w tym planowanie wykonywania zadań), zarządzanie pamięcią, obsługa urządzeń wejścia/wyjścia poprzez sterowniki, zarządzanie plikami oraz mechanizmy ochrony i izolacji (uprawnienia, kontrola dostępu). W wielu systemach istotną rolę odgrywa także wbudowany stos sieciowy. (Źródło: os-book.com)

W systemach przeznaczonych do pracy „biurkowej" często spotyka się środowiska graficzne ułatwiające obsługę. W innych klasach systemów (np. serwerowych lub wbudowanych) administracja bywa prowadzona głównie zdalnie lub z użyciem interfejsu tekstowego. (Źródło: os.ecci.ucr.ac.cr)

Nie ma precyzyjnego określenia, które składniki wchodzą w skład systemu operacyjnego jako jego części. Najczęściej akceptuje się definicję „marketingową", zgodnie z którą to wszystko, co producent udostępnia w ramach zbioru oprogramowania nazywanego systemem operacyjnym, stanowi jego część. Lecz że tak pojemna definicja nie jest zawsze najlepszym wyborem, przekonała się np. firma Microsoft. W latach 90. i na początku XXI wieku integracja wybranych składników (np. Internet Explorer) z systemem Microsoft Windows stała się jednym z elementów sporów antymonopolowych; sprawa w USA zakończyła się ugodą i tzw. Final Judgment (2001–2002). (Źródło: justice.gov)

W ogólnym przypadku w strukturze systemu operacyjnego wyróżnia się jądro oraz programy systemowe, które dostarczane są razem z systemem operacyjnym, ale nie stanowią integralnej części jądra. Jądro jest zbiorem modułów, które ukrywają szczegóły sprzętowej realizacji systemu komputerowego, udostępniając pewien zestaw usług, wykorzystywanych między innymi do implementacji programów systemowych. W dalszej części system operacyjny będzie rozumiany głównie jako jądro, ewentualnie inne elementy oprogramowania integralnie związane z funkcjonowaniem jądra.

Z punktu widzenia kontaktu z użytkownikiem istotny jest interpreter poleceń (shell), który w większości współczesnych systemów jest zwykłym programem działającym w przestrzeni użytkownika (osobnym procesem). Jądro uruchamia go i obsługuje mechanizmy potrzebne do tworzenia kolejnych procesów, ale sam interpreter nie musi być częścią jądra (tak jest m.in. w systemach typu UNIX). Interpreter wykonuje pewne polecenia wewnętrznie, tzn. moduł lub program interpretera dostarcza implementacji tych poleceń. Jeśli interpreter nie może wykonać wewnętrznie jakiegoś polecenia, uruchamia odpowiedni program (tzw. polecenie zewnętrzne), jako odrębny proces. (Źródło: codex.cs.yale.edu)

Nie będziemy zagłębiali się zbytnio w strukturę systemu operacyjnego - będziecie mieli tę tematykę rozwiniętą w przedmiocie poświęconym tylko systemom. Natomiast postaramy się przedstawić jego funkcjonalność oraz właściwości użytkowe.

1.1. Funkcje systemu komputerowego

Komputer bez oprogramowania jest jedynie bezużyteczną skrzynką. Aby tchnąć w niego życie trzeba dodać do niego myśl człowieka zawartą w oprogramowaniu (co nie znaczy, że w konstrukcji sprzętu tej myśli nie ma zawartej). Dopiero połączenie tych dwóch elementów – sprzętu (ang. *hardware*) i oprogramowania (ang. *software*) – tworzy środowisko, w którym można wykonywać różnorodne zadania.

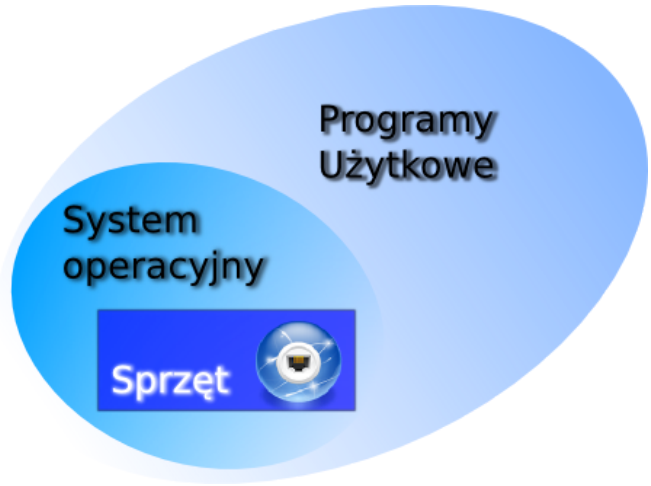
Definicja

Sprzęt i oprogramowanie tworzą *system komputerowy*.



Funkcje systemu operacyjnego można rozpatrywać z różnych punktów widzenia. Lecz zanim przejdziemy do próby wyodrębnienia najważniejszych funkcji, zdefiniujmy główne zadania dla systemu operacyjnego:

- **zarządzanie zasobami systemu komputerowego**. System operacyjny pośredniczy pomiędzy użytkownikiem a sprzętem, dostarczając wygodne środowisko do wykonywania programów. Użytkownik końcowy korzysta z programów (aplikacji), na potrzeby których przydzielane są zasoby systemu komputerowego. Przydziałem tym zarządza system operacyjny, dzięki czemu można uzyskać stosunkowo duży stopień niezależności programów od konkretnego sprzętu oraz odpowiedni poziom bezpieczeństwa i sprawności działania;
- **stworzenie środowiska wygodnego dla użytkownika**. Z tego punktu widzenia zależy nam, aby jak najbardziej odseparować użytkownika od technicznych aspektów sprzętu komputerowego i ukryć przed nim zasadę jego działania (sami możecie ocenić na ile się to współcześnie udało). Przy czym jako użytkownika systemu operacyjnego rozumie się tutaj także programistów tworzących dla danego systemu aplikacje.



Współpraca elementów systemu komputerowego

Efektywność zarządzania zasobami oraz wygodny interfejs dla użytkownika są dwoma ogólnymi, niezależnymi celami projektowymi systemów operacyjnych. W świecie systemów typu UNIX przez długi czas dominował interfejs tekstowy, a graficzny interfejs użytkownika upowszechniał się m.in. dzięki X Window System (od 1984 r.) oraz późniejszym środowiskom pulpitu. W przypadku Microsoft Windows wczesne wersje (1.x–3.x) były graficznym środowiskiem uruchamianym na MS-DOS - i z tego względu nie zasługiwały na miano pełnoprawnego systemu operacyjnego. Linia Windows NT (od 1993 r.) była projektowana jako nowoczesny, 32-bitowy system operacyjny - i to ona jest podstawą współczesnego Windows (computerhistory.org, computerhistory.org, X Window System)

Zarządzanie zasobami

System operacyjny zarządza pracą wszystkich elementów wchodzących w skład systemu komputerowego. Wszystkie te elementy mają określone zasoby.

Formalne pojęcie zasobu wprowadzimy wraz z pojęciem procesu. Na razie zasób będzie rozumiany intuicyjnie jako element systemu komputerowego istotny, czy wręcz kluczowy dla realizacji przetwarzania danych. Tak więc zasobem dla systemu operacyjnego jest np.: czas procesora, pamięć operacyjna, dostęp do urządzeń wejściowych i wyjściowych, dostęp do pamięci masowej. Zazwyczaj funkcja zarządzania zasobami nie jest bezpośrednio wykorzystywana przez użytkownika (czasami nie jest przez niego w ogóle dostrzegana) - wykorzystują ją programy użytkowe (aplikacje). Celem tej funkcji jest optymalizacja wykorzystania zasobów przez użytkowników.

W ramach zarządzania ogólnie rozumianymi zasobami możemy wyróżnić następujące operacje:

- **przydział zasobów**: realizacja żądań dostępu do zasobów w taki sposób, że zasoby używane są zgodnie z intencją użytkowników (np. zagwarantowanie wyłącznego dostępu drukarki, żeby wydruk mojego wielostronicowego dokumentu nie był przetykany stronami Excela kolegi);
- **planowanie dostępu do zasobów**: strategia przydziału zasobów gwarantująca bezpieczeństwo, żywotność, brak zakleszczenia, sprawiedliwość oraz optymalność ich wykorzystania. Zwróćcie uwagę na odróżnienie planowania od samego przydziału — przydział oznacza powiązanie zasobu z realizowanym zadaniem, podczas gdy planowanie wiąże się z podejmowaniem decyzji odnośnie wyboru zadania, któremu zasób będzie przydzielony. I może się zdarzyć taka sytuacja, że ze względu na brak wolnej pamięci jakiś program nie zostanie uruchomiony;
- **ochrona i autoryzacja dostępu do zasobów**: dopuszczanie możliwości użytkowania zasobu tylko przez osoby (procesy) uprawnione i w zakresie przydzielonych im uprawnień. Przez to należy rozumieć zarówno nadanie praw do druku tylko poszczególnym osobom, jak i np. zabronienie programowi A modyfikacji tego obszaru pamięci, który został przydzielony programowi B (obejście tego zabezpieczenia jest najczęściej celem hackerów);
- **odzyskiwanie zasobów**: dołączanie zwolnionych zasobów do zbioru zasobów wolnych po zakończeniu ich użytkowania. Po zakończeniu pracy aplikacji użytkownika, cała przydzielona jej pamięć jest z powrotem zwracana do wspólnej puli i może być przydzielona innej aplikacji;
- **rozliczanie**: rejestrowanie i udostępnianie informacji o wykorzystaniu zasobów w celach kontrolnych i rozrachunkowych.

Brzmi to jak tekst ekonomisty ... trochę w tym prawdy. System operacyjny jest więc Centralnym Ekonomistą Komputera 😊 .

Typowymi zasobami zarządzanymi przez system operacyjny są: procesor, pamięć operacyjna, pamięć masowa (system plików) i urządzenia wejścia-wyjścia. Zarządzanie każdym z nich wymaga specyficznego podejścia.

Procesor jest zasobem współdzielonym przez wiele procesów (wiele programów pracujących razem), w związku z czym zadaniem systemu operacyjnego jest przydział kwantu czasu procesora i wywłaszczanie zadania, które:

- wykorzystało już swój czas;
- nie może kontynuować przetwarzania ze względu na brak innych zasobów (np. brak gotowości urządzeń wejścia-wyjścia);
- ustępuje miejsca zadaniu o wyższym priorytecie (gdy takie zadanie stanie się gotowe do wykonania).

Wspomniane wywłaszczanie polega na zatrzymaniu działania programu: przeniesieniu go w swego rodzaju "stan uśpienia" oraz "obudzeniu" innego, czekającego w kolejce na swój cykl. Z tego punktu widzenia zarządzanie czasem procesora jest zadaniem **cyklicznym**.

Pamięć jest zasobem, który przydzielany jest na wyłączność danego przetwarzania. Zadaniem systemu jest zatem utrzymywanie informacji o zajętości przestrzeni adresowej, znajdowanie i przydzielanie odpowiednich fragmentów wolnej pamięci na żądanie aplikacji użytkownika lub innych modułów systemu operacyjnego, jak również reagowanie na naruszenie ochrony pamięci. W przeciwieństwie do zarządzania czasem procesora, zarządzanie pamięcią wygląda inaczej: system operacyjny nie może dowolnie "odebrać" procesowi całej przydzielonej mu przestrzeni adresowej bez wpływu na jego działanie. Może jednak ograniczać ilość pamięci rezydentnej w RAM, przenosząc rzadziej używane strony do przestrzeni wymiany (paging/swapping) i w ten sposób odzyskiwać wolne ramki pamięci. Tak więc przydział pamięci nie może być uznawany za zadanie cykliczne. (Źródło: docs.redhat.com)

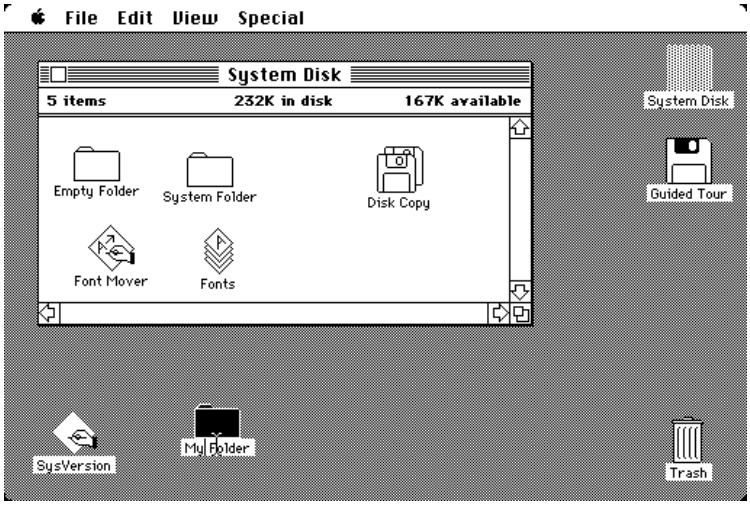
Oprócz prostego przydzielania czasu procesora czy też pamięci, system operacyjny wykonuje także inne zadania powiązane z zarządzaniem zasobami:

- udostępnia środowisko do wykonywania programów : system operacyjny dostarcza struktur danych do organizacji wykonywania programu oraz zachowywania i odtwarzania jego stanu. System operacyjny udostępnia też programistom mechanizmy komunikacji pomiędzy programami (kolejki komunikatów, strumienie, pamięć współdzielona) oraz mechanizmy synchronizacji procesów (np. semafory);
- steruje urządzeniami wejścia-wyjścia: odpowiednie moduły sterujące, integrowane z systemem operacyjnym, inicjalizują pracę urządzeń zewnętrznych oraz pośredniczą w efektywnym przekazywaniu danych pomiędzy jednostką centralną a tymi urządzeniami;
- gwarantuje obsługę podstawowych błędów: system operacyjny reaguje na błędy użytkowników (np. niedostępność zasobów, brak prawa dostępu), programistów (np. błąd dzielenia przez 0, naruszenie ochrony pamięci, nieprawidłowy kod rozkazu) oraz systemu (np. błąd braku strony, błąd magistrali);
- ukrywa przed aplikacjami skomplikowaną budowę sprzętu komputerowego oraz różnice pomiędzy poszczególnymi urządzeniami przez tworzenie abstrakcji, np.:
    - zbiory zapisanych klastrów na dysku widziane są jako pliki o symbolicznych nazwach;
    - każda karta graficzna może być przez programistę traktowana jako dwuwymiarowa tablica pikseli, którym można nadać jedną z 16 milionów barw (niezależnie od trybu graficznego który mamy ustawiony). System operacyjny wraz ze sterownikami i bibliotekami graficznymi zapewnia warstwę pośrednią, dzięki której aplikacja nie musi znać szczegółów konkretnego urządzenia i może zlecać rysowanie w ujednolicony sposób.

Interfejs użytkownika

W początkowych rozwiązaniach systemów operacyjnych interfejs użytkownika był traktowany po macoszemu. Królował terminal tekstowy (inaczej zwany konsolą), w którym można było wpisywać kolejne polecenia dla systemu. I zawsze trzeba było wiedzieć, co należy napisać ... to był problem ;) Dlatego też przez długi okres czasu systemy komputerowe były rzeczywiście trudne w obsłudze i przeznaczone raczej dla dobrze przygotowanych użytkowników. Konsola pozwalała tylko na wprowadzanie i wyświetlanie tekstu, w najbardziej prymitywnych rozwiązaniach wyświetlanie możliwe było tylko linia po linii.

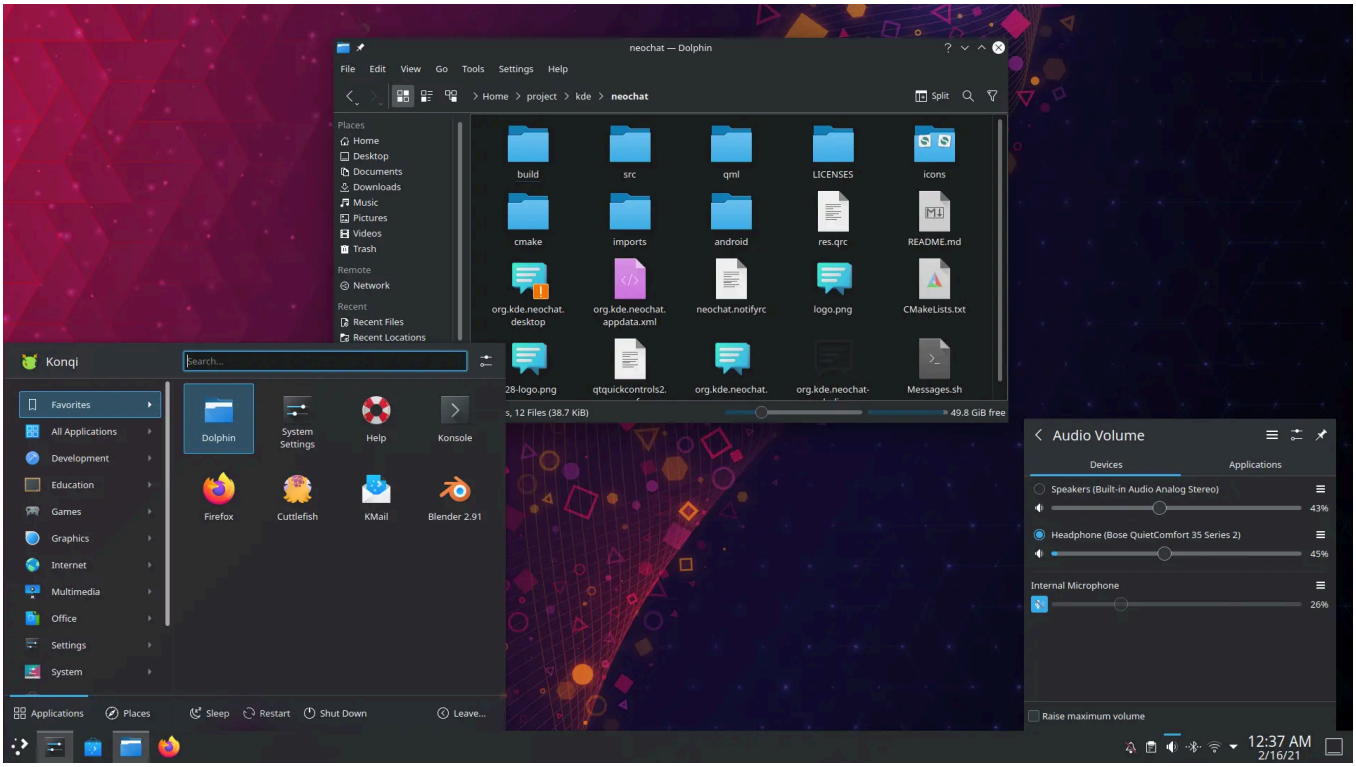
Przełomem, który pozwolił na rzeczywiste zadomowienie się systemów komputerowych "pod strzechami", było pojawienie się graficznych interfejsów użytkownika (ang. *Graphical User Interface*, GUI). W roku 1979 Steve Jobs zwiedzając laboratoria Xerox PARC dostrzegł potencjał opracowanego tam systemu komputerowego z graficznym interfejsem użytkownika (GUI). Xerox nie uznał tego projektu za ważny, natomiast Jobs wkrótce rozpoczął pracę nad nowym systemem z GUI w swojej macierzystej firmie Apple Computers Inc. System ten został nazwany Apple Lisa, nie odniósł jednak sukcesu komercyjnego z racji wysokiej ceny i braku wystarczająco zróżnicowanego oprogramowania. Jobs, straciwszy na skutek wewnętrznych tarć dostęp do Apple Lisa, rozpoczął pracę nad niskobudżetowym komputerem z GUI. Tak powstał Apple Macintosh. Apple Lisa i Apple Macintosh były jednymi z pierwszych komputerów, które szerzej spopularyzowały graficzny interfejs w segmencie komputerów osobistych, choć wiele kluczowych idei GUI rozwijano wcześniej m.in. w Xerox PARC (Alto) oraz w komercyjnym Xerox Star. Upowszechnione przez nie takie standardy graficznego interfejsu jak okna, rozwijalne menu, kursor myszy, kosz na niepotrzebne pliki stały się od tej pory podstawą współczesnych systemów operacyjnych. (Źródło: web.stanford.edu, computerhistory.org, Xerox Alto)



Jeden z pierwszych upowszechnionych na szeroką skalę GUI — klasyczny system Macintosh (System 1).

Środowisko graficzne jest grupą wzajemnie współpracujących programów (części systemu operacyjnego), zapewniającą możliwość wykonywania podstawowych operacji na komputerze (takich jak uruchamianie programów, poruszanie się po katalogach, zmiana konfiguracji systemu itp.) w trybie graficznym, najczęściej okienkowym. Zapewnia alternatywny dla konsoli sposób pracy na komputerze.

Najważniejszym elementem graficznego interfejsu jest okno programu (lub kilka takich okien, choć takie konfiguracje są niezbyt lubiane przez użytkowników). Wewnątrz takiego okna są rozmieszczone elementy interakcyjne, zwane widgetami (lub "kontrolkami", nawiązując do pulpitów sterowniczych). Użytkownik komunikuje się z aplikacją pośrednio przez te widgety najczęściej za pomocą myszy i klawiatury. Mysz jest odpowiedzialna za przesuwanie tzw. kursora myszy, wskazującego odpowiednią pozycję na ekranie, a naciskanie przycisków jest związane z obszarem, w którym zawiera się aktualna pozycja kursora. Klawiatura jest związana z kolei z pojęciem tzw. "skupienia". Skupienie jest stanem, który może posiadać w jednym czasie dokładnie jeden widget w całym systemie okienkowym. Jeśli użytkownik naciska klawisze na klawiaturze, to wtedy informacja o naciskaniu klawiszy przekazywana jest do tego widgetu, który aktualnie "posiada skupienie" lub "jest w stanie skupienia".



KDE Plasma — jedno z najpopularniejszych środowisk pulpitu w systemach Linux i innych systemach uniksowych.

1.2. Budowa systemu operacyjnego

Obarczyliśmy więc system operacyjny zadaniami. Z lektury poprzedniego rozdziału wynika, że jest ich niemało ... podobnie jak niemała jest liczba możliwych podzespołów sprzętowych, które należy obsłużyć. Z tego względu system operacyjny jest dziś najczęściej systemem dużym i złożonym. To skomplikowany twór, który musi zapewniać z jednej strony dużą niezawodność i stabilność działania, a z drugiej łatwość rozbudowy i modyfikacji (na przykład dostosowania do szybko zmieniających się rozwiązań sprzętowych).

Dlatego też w systemach ogólnego przeznaczenia rzadko spotyka się dziś jądra w w pełni statycznej, „czysto monolitycznej" postaci. Najczęściej są to jądra monolityczne, ale modułowe (z możliwością ładowania sterowników i innych elementów w postaci modułów). (Przykład: dokumentacja Red Hat o modułach jądra)

Model warstwowy

W takim przypadku możemy rozrysować sobie komputer w następujący sposób:



Model warstwowy systemu komputerowego

Na samym dole mamy sprzęt, potem kolejne warstwy abstrakcji, z których każda wykorzystuje możliwości tej, która jest poniżej i udostępnia swoją funkcjonalność wyżej, stanowiąc jednocześnie pewnego rodzaju granicę - ukrywając niepotrzebne szczegóły przed warstwami wyższymi.

Mały przykład: rozważmy w jaki sposób obsługiwany jest dysk twardy. Jak wiecie z poprzednich lekcji, dysk twardy jest to zespół okrągłych talerzy pokrytych substancją magnetyczną, nad którymi poruszają się głowice. Tylko jak je zmusić do ruchu? W warstwie fizycznej realizują to mikrosilniki krokowe. Lecz z punktu widzenia systemu operacyjnego jest to zbyt skomplikowane. Dlatego też producenci dysków twardych wyposażają je w tzw. firmware - oprogramowanie bardzo niskopoziomowe, zaszyte w pamięci ROM urządzenia, które zawiera procedury sterowania silnikami krokowymi. Aby z nich korzystać, nie trzeba znać szczegółów budowy silnika, napięć nim sterujących, itp, itd. W praktyce system operacyjny i sterownik operują dziś zwykle na adresach logicznych bloków (LBA), a mapowanie adresu logicznego na fizyczne położenie danych (z uwzględnieniem remapowań, strefowania itd.) realizuje kontroler/firmware urządzenia. (Źródło: Logical block addressing (LBA))
Tak więc, dla oprogramowania niskopoziomowego (sterownika i jądra) pewne, nieistotne z jego punktu widzenia, szczegóły zostały ukryte poprzez warstwę firmware. Idąc dalej tym krokiem - jądro wykorzystuje procedury firmware wewnątrz swoich procedur, które zapisują logiczne struktury danych (czyli pliki) w fizycznych blokach i sektorach HDD. Z punktu widzenia aplikacji i finalnego użytkownika znów pewne nieistotne szczegóły zostały ukryte. Aby skorzystać z tych procedur, wydajemy proste polecenia - utworzenia, zapisania, skasowania pliku na dysku. Jądro systemu przetworzy te polecenia na ciąg poleceń, a firmware przetworzy je na operacje sterujące fizycznym urządzeniem.

Dostaliśmy więc coś w rodzaju radia (na zewnątrz mamy tylko pokrętła, które pozwalają nam na słuchanie muzyki i nie musimy wiedzieć co się dzieje wewnątrz) - mamy dostęp do plików, które można zapisywać i odczytywać bez konieczności posiadania wiedzy, co się dzieje fizycznie w maszynie nazywanej komputerem. Co więcej - dostaliśmy też abstrakcję - system plików. I z punktu widzenia aplikacji użytkownika nie jest istotne gdzie jest on umieszczony fizycznie. Korzystając z MS Word tak samo zapisujemy plik na pen-drive jak na HDD, mimo że fizyczna zasada działania tych dwóch urządzeń jest kompletnie różna - lecz aplikacja w ogóle o tym nie wie.

Na tym zakończymy omawianie modelu warstwowego. Jest on przydatny w analizie i budowie systemów komputerowych, jednakże jak na potrzeby tego podręcznika - zbyt abstrakcyjny i teoretyczny. W jego miejsce proponujemy przyjąć uproszczony schemat systemu operacyjnego.

Schemat budowy systemu operacyjnego

W podejściu pokazanym poniżej ograniczamy liczbę warstw i jednocześnie pokazujemy która warstwa może korzystać z której:



System operacyjny - części składowe.

W podobny sposób można traktować wszystkie współczesne systemy operacyjne.

- Jądro systemu (występuje w kilku odmianach, opisanych poniżej) obsługuje sprzęt (wraz ze sterownikami działającymi w trybie jądra). Programy w trybie użytkownika korzystają z urządzeń pośrednio – przez wywołania systemowe i sterowniki – dzięki czemu system może kontrolować dostęp i współdzielenie zasobów - patrz: One Windows kernel
- API udostępnia funkcje jądra aplikacjom, w tym powłoce systemowej.
- Programy narzędziowe (w tym powłoka) są pierwszym zewnętrznym "oknem na świat" systemu operacyjnego, z którym może zetknąć się użytkownik.
- Aplikacje użytkownika - w ten sposób wreszcie dotarliśmy do programów, z których korzystamy na co dzień.

Jądro

Większość systemów operacyjnych opiera się o koncepcję jądra, która w naturalny sposób wynika z modelu warstwowego. W takim wypadku jądro jest po prostu nazwą nadaną najniższej warstwie w pełni implementowanej jako program wykonywany przez CPU.

Do typowych funkcji realizowanych przez jądro można zaliczyć:

- zarządzanie sprzętem: przydział czasu procesora, obsługę przerywań, obsługę układów wejścia/wyjścia;

- zarządzanie procesami: kolejkowanie, zapewnianie im wzajemnej komunikacji, synchronizację procesów, tworzenie i destrukcję procesów oraz ich usypianie i wznawianie (więcej o tym będzie w dalszej części podręcznika);
- zarządzanie pamięcią: Jądro jako jedyny program w systemie komputerowym ma nieograniczony dostęp do całej zainstalowanej pamięci. To ono przydziela fragmenty tej pamięci pracującym programom (procesom) oraz tworzy tzw. wirtualną przestrzeń adresową - czyli konwertuje fizyczne adresy pamięci RAM na ich odpowiednik unikalny dla każdego z procesów. W ten sposób możemy zapobiegać próbom odwołań się do tej samej komórki pamięci przez kilka procesów. Co więcej - dzięki wirtualnej przestrzeni adresowej można część danych z RAM (chwilowo nie wykorzystywanych) przenieść np. na HDD, dzięki czemu zwiększa się wirtualnie ilość pamięci operacyjnej dostępnej w systemie operacyjnym;
- zarządzanie urządzeniami: we wzorcowo zbudowanym systemie operacyjnym jedyną metodą dostępu do urządzeń peryferyjnych jest wywołanie odpowiedniej funkcji jądra. Dzięki temu możliwe jest równoczesne korzystanie z tego samego urządzenia przez wiele aplikacji użytkownika - bez wchodzenia sobie nawzajem w drogę. Napisaliśmy o wzorcowym systemie - niestety, spełnienie tego postulatu w pełni nie jest możliwe. Popatrzmy przykładowo na karty graficzne: skoro dostęp do nich miałby być możliwy tylko i wyłącznie poprzez jądro, to jądro powinno znać wszystkie funkcje karty (żeby mogło je wywołać). A skoro jądro ma być uniwersalne, to powinno znać wszystkie funkcje wszystkich kart dostępnych teraz i w przyszłości (sic!) na rynku. Dlatego też, w niektórych obszarach (np. grafika) stosuje się rozbudowany stos sterowników i bibliotek: część działa w trybie użytkownika (np. komponenty ładowane przez runtime), a część w trybie jądra (sterowniki kernel-mode). Dla programisty wygląda to jak korzystanie z API (np. Direct3D), ale pod spodem nadal jest to kontrolowana współpraca warstwy użytkownika i jądra User-mode display drivers.
- realizacja wywołań systemowych - czyli z jednej strony realizacja żądań aplikacji użytkownika, a z drugiej generowanie żądań do programów pochodzących od systemu (np. wykonywanie jakichś funkcji co określony czas czy przerysowanie zawartości okienka).

Po uruchomieniu komputera specjalny mały program, nazywany bootloaderem, ładuje jądro systemu operacyjnego do pamięci i przekazuje mu sterowanie maszyną. Jądro uruchamia w tym momencie pierwszy, główny proces, inicjalizuje wszystkie urządzenia peryferyjne i przechodzi w stan oczekiwania na zdarzenia (np. ruchy myszą, żądania wydruku od aplikacji, przerwania, itp.) W czasie, gdy nie pojawiają się żadne zdarzenia, uruchamiane są wątki bezczynności (idle threads). W Windows jest to widoczne jako *System Idle Process*, który reprezentuje niewykorzystany czas CPU (zwykle jeden wątek bezczynności na rdzeń / logiczny procesor). (Źródło: System Idle Process) W trybie jądra (kernel mode) każda operacja dozwolona przez architekturę procesora może być wykonana (dowolna instrukcja, dowolna operacja wejścia/wyjścia, zapis lub odczyt dowolnego miejsca w pamięci). W trybie użytkownika niektóre z tych operacji są zabronione, przy czym wykonania niedozwolonej operacji w tym trybie zapobiega sprzętowe ograniczenie wbudowane w procesor.

Kod może wykonywać się w dwóch trybach: użytkownika oraz jądra. Typowo aplikacje działają w trybie użytkownika, a do trybu jądra przechodzi się m.in. podczas wywołań systemowych, obsługi przerwań i niektórych wyjątków. Błąd w kodzie wykonywanym w trybie jądra może unieruchomić cały system komputerowy.

Opracowanie jądra jest jednym z najtrudniejszych i najbardziej skomplikowanych zadań programistycznych. Jako główny, najważniejszy program w systemie komputerowym, jądro powinno spełniać różne (często przeciwstawne) wymagania. Musi cechować się wysoką wydajnością, stabilnością i odpornością na błędy aplikacji oraz awarie podzespołów. Dodatkowo, często nie może wykorzystywać mechanizmów abstrakcji, które samo udostępnia innym programom, czyli np. przy jego tworzeniu nie można zapomnieć o fizycznej realizacji zapisu na dysk i opierać się jedynie na koncepcji plików. Tak więc jądro samo w sobie stanowi całkiem skomplikowany system informatyczny, który może być zbudowany w różny sposób. Wyróżniamy kilka podstawowych metod konstrukcji jąder

Jądro monolityczne

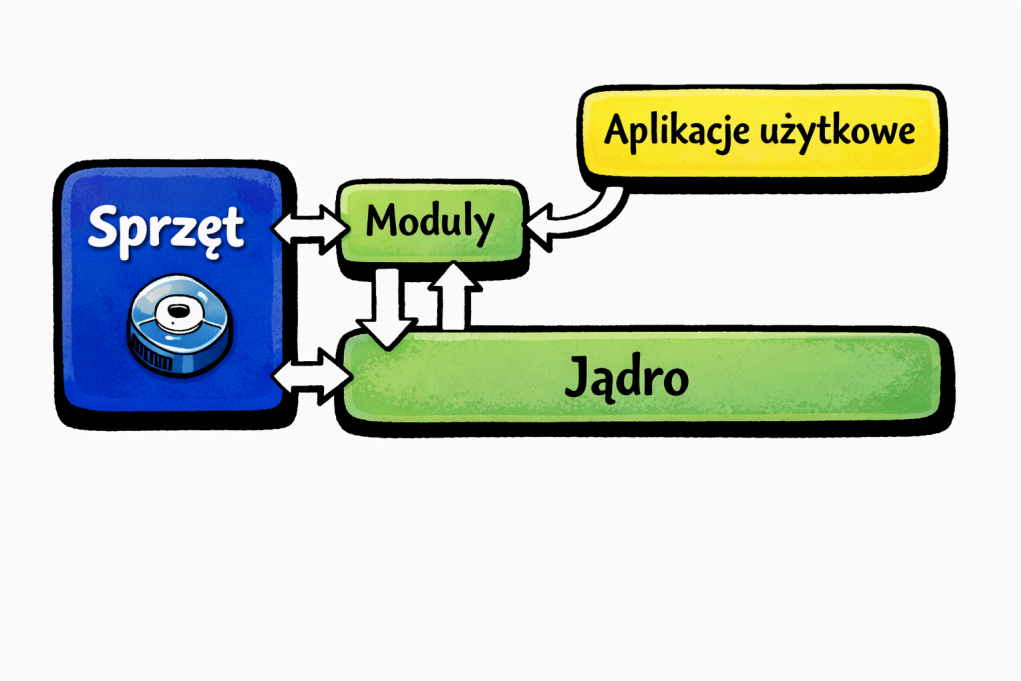
Często stosowane w systemach Unix-owych. Wszystkie zadania są wykonywane przez jądro, będące jednym, dużym programem pracującym w pełni (wszystkie funkcje) w trybie jądra. Zaletami tej techniki są: prostota, stabilność oraz łatwość komunikacji pomiędzy różnymi członami jądra (to przecież w tym wypadku jeden program!). Dodatkowo, systemy z jądrem monolitycznym zazwyczaj działają szybciej (szczególnie w komputerach o małej liczbie CPU) niż ich główna alternatywa - mikrojądro. Główna wada natomiast wynika z problemów w rozwijaniu takiego jądra (ze względu na jego wielkość) oraz dodawaniu do niego procedur obsługi nowych urządzeń peryferyjnych. Dodanie obsługi nowego urządzenia wymaga kompilacji całego jądra, co zajmuje czas i jest trudne do wykonania przez nawet zaawansowanego użytkownika.

System operacyjny z monolitycznym jądrem.

W czystej postaci jądro monolityczne występuje tylko w zastosowaniach niszowych, natomiast w dużej mierze takie rozwiązanie stosują systemy Linux, Solaris, OpenBSD, FreeBSD czy Mac OS-X - jądra tych systemów umożliwiają ładowanie modułów, co jest sprzeczne z koncepcją jednego dużego programu, ale daje możliwość stosowania driverów, czyli dodawania obsługi nowego sprzętu bez konieczności ponownej kompilacji całego jądra.

Mikrojądro

W tej technice z monolitycznego jądra zostaje tylko jego podstawowa część (odpowiedzialna za zarządzanie procesami oraz pamięcią). Części odpowiedzialne za bardziej wyrafinowane funkcje (m.in. obsługę urządzeń peryferyjnych) są wydzielone do funkcjonalnych bloków albo realizowane jako zwykłe procesy w trybie użytkownika. W większości przypadków takie bloki funkcjonalne mogą być ładowane i resetowane nie przerywając pracy systemu komputerowego, ponadto zwiększanie funkcjonalności jądra (i w konsekwencji - całego systemu) jest możliwe bez wprowadzania jakichkolwiek zmian w jego podstawowej części. Główną wadą takiego podejścia jest wolniejsza praca systemu.



System operacyjny z mikrojądrem.

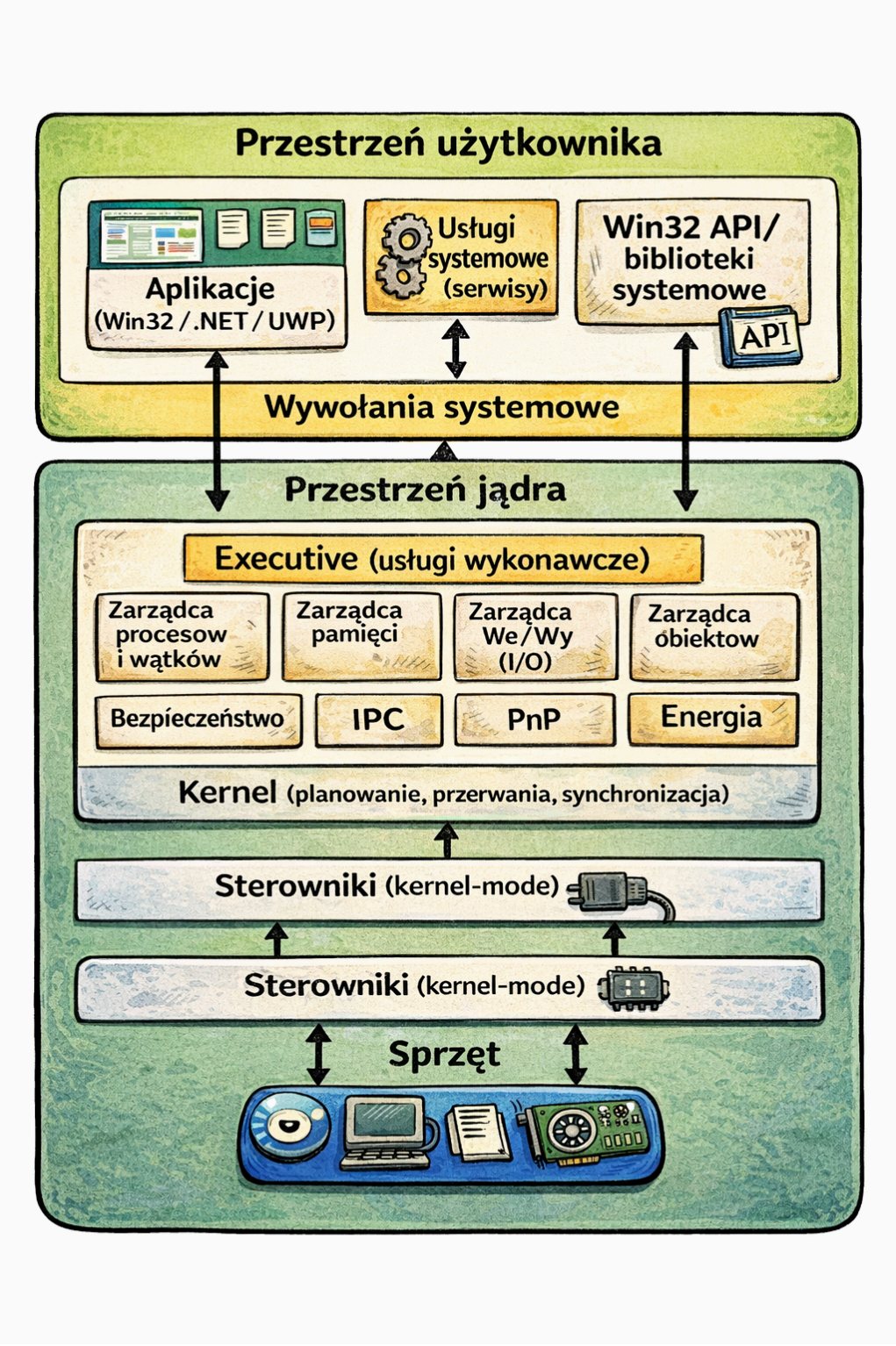
Dobrymi przykładami systemów operacyjnych opartych na mikrojądrze są AmigaOS, Amoeba, QNX, Minix, czy GNU Hurd, mikrojądrami są także jądra znane jako Mach i L4. Firma Microsoft (Microsoft Research) prowadziła projekt badawczy Singularity, w którym eksperymentowano z innym podejściem do konstrukcji systemu (m.in. silniejszymi gwarancjami izolacji). Był to jednak projekt badawczy opisywany w publikacjach, a nie „typowy” produktowy system operacyjny. (Źródło: publikacja o Singularity)

Jądro hybrydowe

Jest to kompromis między architekturą jądra monolitycznego i mikrojądra. W krytycznych usługach - np. stos sieci - usługi są na stałe wkompilowane w główny kod jądra, inne usługi pozostają oddzielone od głównego jądra i działają jako serwery (w przestrzeni jądra). Dzięki temu rozwiązaniu możliwe jest zachowanie wydajności jądra monolitycznego dla kluczowych usług.

Wydzielenie takiego rodzaju jądra budzi kontrowersje wśród specjalistów. Wielu z nich uważa, iż w porównaniu do podobnego jądra monolitycznego, pojęcie to jest niczym innym jak dobrym marketingiem.

Pomysłem stojącym za tym nowym pseudo-rodzajem jądra jest struktura podobna do mikrojądra, ale zaimplementowana jak jądro monolityczne. W przeciwieństwie do mikrojądra, wszystkie (lub prawie wszystkie) usługi wykonywane są w przestrzeni jądra. Podobnie jak w jądrze monolitycznym, nie ma stratjaw wydajności wywołanych przepływem komunikatów mikrojądra. Jednakże, podobnie jak w jądrach monolitycznych, nie ma korzyści wynikających z umieszczenia usług w przestrzeni użytkownika.



Budowa jądra systemu Windows

Tego typu jądro jest podstawą systemów Windows od NT do współczesnej wersji 11. Jest ono wykorzystywane również przez mniej popularne systemy, jak np. BeOS.

Inne typy jąder

Oprócz omówionych, istnieją także rozwiązania które nie zdobyły większej popularności, jednakże dla pełnego przekazu wymienimy je tutaj:

- nanojądro - tylko bardzo mała część operacji wykonywana jest w trybie kernela. Czasem tylko warstwa dostępu do urządzeń fizycznych wykonywana jest w tym trybie w celu dostarczenia funkcjonalności systemów czasu rzeczywistego do normalnych systemów operacyjnych. Przykładowa implementacja - Adeos;
- exojądro - minimalistyczny kernel umożliwiający uruchomienie tzw. maszyn wirtualnych, z których każda jest kopią fizycznego systemu z ograniczonymi zasobami. Rolą jądra jest odpowiedni przydział tych zasobów do poszczególnej wirtualnej maszyny i pośredniczenie wymianie informacji między nimi a fizycznymi urządzeniami. Przykładowe implementacje - Nemesis, ExOS.

API (*Application Programming Interface*)

Usługi jądra nigdy nie są bezpośrednio widoczne dla użytkownika czy aplikacji. Użytkownik może je wykorzystywać i wywoływać korzystając z powłoki systemu, natomiast aplikacje najczęściej wykorzystują tzw. API - zbiór poleceń, które aplikacje użytkownika mogą wydawać systemowi operacyjnemu (np.: „zamknij plik", „odczytaj znak z klawiatury"). Warstwa API (ang. *Application Programming Interface*) pełni rolę pośrednika między systemem operacyjnym a programami użytkownika. API wykorzystywane jest jedynie przez programistów i zazwyczaj dostarczane w postaci dokumentacji oraz zestawów dla programistów (np. plików nagłówkowych C/C++, bibliotek i narzędzi). Kompletne API dużego systemu operacyjnego to bardzo rozbudowany zestaw funkcji i interfejsów, a jego szczegóły różnią się pomiędzy systemami operacyjnymi.

Programy narzędziowe

Najważniejszym z nich jest **powłoka systemu**, czyli jego interfejs widoczny dla użytkownika. To ona decyduje o wyglądzie i funkcjonalności systemu operacyjnego dla każdego z Was.

Dwa najczęściej spotykane typy powłoki to:

- tekstowy interpreter poleceń, np. BASH (*Bourne-Again Shell*) w Unixach czy CLI (*Command Line Interpreter*) w Windows. Tego typu powłokę najczęściej wykorzystują bardziej zaawansowani użytkownicy;
- graficzny interfejs użytkownika (GUI), czyli np. Explorer w Windows, czy też KDE czy Gnome dla Unixów. Graficzne interfejsy użytkownika to jest ten typ powłoki, z którym na co dzień styka się zdecydowana większość użytkowników komputerów.

Powłoka często sama zawiera podstawowe polecenia, gdy jednak wydane przez użytkownika polecenie nie jest wbudowane, uruchamiany jest zewnętrzny program narzędziowy, np. do wyświetlenia zawartości aktualnego katalogu w powłoce tekstowej wywoływany jest program nazwany **dir** dla Windows oraz **ls** dla Unix-ów.

Z technicznego punktu widzenia powłoka nie różni się od innych programów narzędziowych, czy też nawet aplikacji zewnętrznych. Jest normalnym programem, pracującym w trybie użytkownika i może być zmieniona przez inny program definiowany przez użytkownika, co jest wykorzystywane do tworzenia rozmaitych nakładek, podmieniających np. interfejs Windows na podobny do MacOS.

1.3. Klasyfikacja systemów operacyjnych

Systemy operacyjne w czasie swego istnienia przeszły znaczące zmiany. Pierwsze z nich były bardzo proste i mało wydajne. Dzisiejsze systemy operacyjne są systemami o dużym stopniu skomplikowania, zajmują nieraz wiele gigabajtów przestrzeni dyskowej, a w pamięci RAM wykorzystują znaczące zasoby zależnie od konfiguracji i uruchomionych usług. Ich możliwości są niewyobrażalne w porównaniu z możliwościami ich odpowiedników sprzed lat.

Systemy wsadowe

Systemy wsadowe były jednymi z pierwszych systemów operacyjnych. Ich konstrukcja była bardzo prosta, a co za tym idzie również i ich możliwości były niewielkie.

Były one instalowane w pierwszych komputerach, w których urządzeniami wejściowymi były czytniki kart perforowanych. Użytkownik takiego systemu przygotowywał najpierw program i zapisywał go na kartach. Tak przygotowane dane trafiały do operatora komputera, który grupował podobne zadania w tak zwane wsady (ang. *batch*) i wprowadzał do komputera. Po przetworzeniu przez jednostkę centralną wyniki były wyprowadzane na drukarkę wierszową.

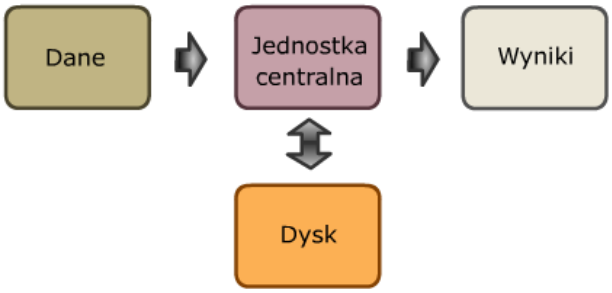
Zasadniczą cechą wsadowego systemu operacyjnego było:

- brak kontroli użytkownika nad wykonywanymi zadaniami;
- wykonywanie przez jednostkę centralną tylko jednego zadania w danym czasie;
- bardzo długie okresy bezczynności jednostki centralnej, a co za tym idzie niska wydajność.

Niskie wykorzystanie jednostki centralnej wynikało przede wszystkim z przestojów spowodowanych niską prędkością urządzeń wejścia-wyjścia.

Aby podnieść wydajność takich systemów wprowadzono mechanizm, któremu nadano nazwę *spooling*.

Spooling wykorzystuje technologie dyskowe i polega na buforowaniu danych na dyskach przed i po ich wprowadzeniu do pamięci operacyjnej. Umożliwiało to znaczne zwiększenie wydajności systemu, rzadziej procesor musiał oczekiwać na dane od użytkownika podawane poprzez np. czytnik kart dziurkowanych... ;)



Mechanizm spoolingu

Spooling umożliwiał jednoczesne przetwarzanie danych przez jednostkę centralną i pracę urządzeń wejścia-wyjścia. Mechanizm ten jest wykorzystywany do dziś — najbardziej widocznym przykładem jest buforowanie i kolejkowanie zadań wydruku przez usługę spoolera drukowania. (Źródło: Microsoft: Print Spooler)

Systemy wieloprogramowe

Zastosowanie buforowania danych na dysku (lub aktualnie w pamięci operacyjnej) dało systemowi nowe możliwości. Nie musiał on już teraz wykonywać zadań w kolejności, w jakiej zostały one wprowadzone do systemu. Otrzymał prawo wyboru 😊 System operacyjny dysponując listą zadań, do której miał łatwy dostęp, rozpoczął pracę koncepcyjną polegającą na *planowaniu zadań.* Zadania były wybierane do realizacji w sposób maksymalizujący wydajność systemu. Odbywa się to mniej więcej tak...

Wybrane zadania są umieszczane w pamięci operacyjnej.

Jednostka centralna rozpoczyna pracę nad jednym zadaniem. Po pewnym czasie może okazać się, że wykonywane zadanie zażądało dostępu do dysku lub czytnika taśm. W systemie jednozadaniowym jednostka centralna musiała czekać na dostęp do dysku lub zamontowanie czytnika. W systemie wieloprogramowym (i później także w systemie z podziałem czasu) system operacyjny może odłożyć takie zadanie i pobrać do wykonania następne, znajdujące się w pamięci operacyjnej. Tego typu zachowanie zwiększa oczywiście wydajność systemu, ale wymusza konieczność uporania się z wieloma trudnymi problemami:

- w jaki sposób wybierać zadania do wykonania?
- jak zapewnić ochronę danych umieszczonych w pamięci?
- w jaki sposób i kiedy współpracować z pamięcią dyskową?

i tym podobne.

Systemy wieloprogramowe nie rozwiązały wszystkich problemów. W dalszym ciągu użytkownik nie miał możliwości ingerencji w wykonywany program w trakcie jego trwania. Musiał więc przewidzieć możliwe zachowania programu przed jego uruchomieniem. Trudną sprawą było też testowanie programów, gdyż nie można było zmienić jego przebiegu obserwując jego zachowanie.

Systemy z podziałem czasu

Systemy z podziałem czasu wykorzystują wielozadaniowość (ang. *multitasking*), ale w praktyce wielozadaniowość występuje też w innych klasach systemów. W tej sekcji chodzi o taki typ wielozadaniowości, w którym celem jest interaktywna praca użytkowników i krótki czas odpowiedzi.

Systemy te dzielą czas procesora między zadania umieszczone w pamięci. Jednostka centralna wykonuje więc jedno zadanie przez pewien czas, a następnie odkłada je i zabiera się za następne. Procesor przełącza się więc nieustannie między zadaniami. Te przełączenia występują na tyle szybko, że użytkownik może pracować z każdym z wykonywanych programów nie zauważając tych przełączeń.

Dzięki temu użytkownik systemu komputerowego ma możliwość interakcji z systemem. Może też wydawać polecenia programowi i oglądać w trakcie jego wykonywania wyniki działania programu.

Aby w sposób wydajny wykorzystywać mechanizm wielozadaniowości system operacyjny powinien charakteryzować się następującymi cechami:

- bezpośrednim dostępem do systemu plików;
- krótkim czasem odpowiedzi;
- mechanizmami decyzyjnymi;
- mechanizmami zarządzania pamięcią;
- mechanizmami ochrony zadań;
- mechanizmami współpracy z dyskiem;
- mechanizmami synchronizacji zadań;
- mechanizmami komunikacji między zadaniami.

Systemy równoległe

Systemy równoległe są przystosowane do obsługi systemów komputerowych wyposażonych w więcej niż jeden procesor lub wiele rdzeni. W praktyce spotyka się architektury SMP (współdzielona pamięć) oraz NUMA (pamięć fizycznie rozproszona, ale dostępna dla wszystkich procesorów/rdzeni z różnymi opóźnieniami). Dzięki zastosowaniu systemów wyposażonych w kilka procesorów można zwiększyć wydajność całego systemu, zmniejszyć jego koszty poprzez wykorzystanie wspólnych urządzeń zewnętrznych, obudów, zasilaczy itp. oraz podnieść niezawodność dzięki dublowaniu zadań.

Systemy jedno- i wielo- dostępowe

Systemy operacyjne można podzielić również biorąc pod uwagę liczbę użytkowników, których system potrafi obsłużyć w jednym czasie. Starsze systemy operacyjne oraz niektóre rozwiązania typowe dla komputerów osobistych przeznaczone były i są dla jednego użytkownika interaktywnego. W systemach tych wszystkie zasoby komputera dostępne są w praktyce dla jednej osoby pracującej „przy konsoli", a mechanizmy autoryzacji i ochrony danych bywają ograniczone lub uproszczone.

Wiele współczesnych systemów dla komputerów osobistych obsługuje jednak wielu użytkowników (konta, uprawnienia, profile), ale zwykle zakłada jednego użytkownika interaktywnego naraz na danej maszynie. Przykładem są klienckie wersje Windows, które wspierają wiele kont i szybkie przełączanie użytkowników, ale nie dają możliwości równoległej pracy dwóch lub więcej użytkowników. (Źródło: Microsoft: Fast User Switching)

Na drugim biegunie mamy do czynienia z systemami wielodostępowymi. W tym przypadku na jednym komputerze może jednocześnie pracować wiele osób, albo bezpośrednio korzystając z lokalnych urządzeń wejścia/wyjścia, lub zdalnie - z poziomu terminala czy też innego systemu komputerowego. Przykładem takiego podejścia jest Linux, a w świecie Windows — systemy serwerowe z uruchomionymi usługami Remote Desktop Services (dawniej Terminal Services). (Źródło: Microsoft: Remote Desktop Services

Systemy rozproszone

Systemy rozproszone przeznaczone są do obsługi komputerów, w których wykonywane zadania są rozdzielone między kilka procesorów. W odróżnieniu jednak od systemów równoległych, procesory te nie dzielą wspólnej szyny danych ani zegara. Procesory komunikują się ze sobą za pomocą szybkiej specjalizowanej sieci lokalnej, lub szybkich łączy dalekiego zasięgu.

Dzięki zastosowaniu systemów rozproszonych można łączyć ze sobą komputery słabsze i lepsze, wykorzystywać drukarki zdalne, stosować rozproszone bazy danych itp. Kolejną zaletą systemów rozproszonych jest możliwość przyspieszenia obliczeń. Dzieje się tak, jeśli dużą liczbę obliczeń można podzielić na kilka współbieżnych procesów. Użytkownik może wykonywać nawet skomplikowane obliczenia siedząc przy mniej wydajnej maszynie, jeśli część zadań przejmie procesor z bardzo silnej i wydajnej jednostki zdalnej.

Systemy klastrowe

System klastrowy jest pojęciem zawężającym pojęcie systemu rozproszonego. Kontroluje on pracę komputerów, które stanowią klaster, czyli zbiór powiązanych ze sobą komputerów, którego celem jest poprawienie wydajności, dostępności i odporności na awarię w stosunku do pojedynczego komputera. Najbardziej znaną klasą komputerów klastrowych jest klaster Beowulf będący zbiorem identycznych komputerów PC połączonych siecią Ethernet. Systemem operacyjnym kontrolującym pracę klastrów bywa najczęściej Linux, BSD lub Solaris, wyposażone w dodatkowe oprogramowanie wspomagające zarządzanie węzłami oraz przekazywanie komunikatów i zadań. W środowiskach klastrowych często stosuje się też warstwy do zarządzania zasobami i uruchamiania zadań (systemy kolejkowe). Historycznie istniały gotowe „zestawy" instalacyjne, takie jak Open Source Cluster Application Resources (OSCAR). (Źródło: OSCAR na SourceForge)

Systemy czasu rzeczywistego

Systemy operacyjne czasu rzeczywistego stanowią odrębną grupę systemów operacyjnych. Są to systemy wysoce specjalizowane, które muszą zapewnić określony czas reakcji systemu komputerowego na bodźce zewnętrzne. Tak więc widać, że podstawową cechą odróżniającą takie systemy od pozostałych rozwiązań jest czas. Jednakże nie chodzi tutaj głównie o jak najszybsze działanie systemu (jak się często mylnie sądzi), lecz o pewność, że w możliwym do określenia czasie system wykona zadanie mu powierzone (czyli determinizm czasowy). Istnieją dwa rodzaje systemów RT – soft i hard real time OS. Różnica między tymi systemami polega na tym, że w systemach hard zachowanie systemu (łącznie z czasami i kolejnością wywołań) jest w pełni zdeterminowane, to znaczy zawsze możliwe do określenia. W systemach soft RT wymagania co do determinizmu zdarzeń są nieco obniżone, choć zwykle można określić czas i kolejność wywołań.

Ze względu na rygorystyczne wymagania czasowe muszą one być lepiej dopasowane do sprzętu, na którym działają. Często stosuje się w nich mechanizmy ochrony (np. separację zadań), jednak dobór rozwiązań zależy od wymagań deterministycznych i od platformy sprzętowej.

Nietrywialnym problemem jest w tym przypadku algorytm szeregowania oraz podziału czasu. W systemie operacyjnym czasu rzeczywistego trzeba określić, któremu z procesów należy przydzielić procesor oraz na jak długi czas, aby wszystkie wykonywane procesy spełniały zdefiniowane dla nich ograniczenia czasowe, dlatego też stosuje się w nich specjalizowane, nie omawiane w tym podręczniku, algorytmy szeregowania procesów.

Systemy operacyjne czasu rzeczywistego znajdują zastosowanie głównie w komputerach wykorzystywanych do sterowania - od sterowania ABS w samochodach osobowych, przez przemysłowe układy automatyki, a kończąc na sterowaniu łazikiem marsjańskim czy promem kosmicznym. Inną grupą zastosowań jest sprzęt medyczny, wykorzystywany do nadzorowania życia ludzkiego.

Do przykładowych reprezentantów tego typu systemów należy QNX, VxWorks, LynxOS czy też systemy oparte o jądro Linux skonfigurowane pod kątem czasu rzeczywistego. W świecie Linuksa szeroko używanym podejściem jest PREEMPT_RT (real-time preemption), które zostało włączone do głównej gałęzi jądra Linuksa. (Źródło: SUSE: Real-Time Linux Kernel Integration; oraz PREEMPT_RT)

Systemy operacyjne typu wbudowanego (embedded)

Systemy typu embedded są przeznaczone do kontrolowania prostych urządzeń takich jak palmtopy, telefony, konsole nawigacji GPS itp. Częstokroć systemy te zaprojektowane są podobnie jak systemy czasu rzeczywistego z uwzględnieniem zwykle małej wydajności sprzętowej urządzeń dla których są przeznaczone. Funkcjonalność tych systemów zazwyczaj jest minimalna co pozwala na minimalizację zajmowanej przez nie pamięci operacyjnej i innych zasobów.

Kiedyś ten typ systemów był wyraźnie wydzielony ze względu na wspomnianą niską wydajność urządzeń na których pracował. W dniu dzisiejszym osiągi procesorów ARM montowanych we współczesnych urządzeniach kieszonkowych bywają porównywalne z komputerami osobistymi sprzed kilku lat, w związku z tym najpopularniejsze systemy wbudowane (Android, iOS) pod względem funkcjonalności zbliżają się do typowych systemów operacyjnych.

Przykładowymi systemami operacyjnymi typu wbudowanego są Embedded Linux, Minix3, VxWorks, Android oraz iOS. Historycznie do tej grupy zaliczało się również Windows CE, PalmOS, Windows Mobile oraz Windows Phone / Windows 10 Mobile

1.4. Podsumowanie

System operacyjny kontroluje pracę poszczególnych komponentów systemu komputerowego. Umożliwia poprawną pracę podzespołów komputera oraz komunikację jednostki centralnej z innymi podzespołami. Troszczy się również o prawidłową pracę programów użytkowych i odpowiednie przydzielanie zasobów pamięci operacyjnej. Wreszcie kontroluje procesy zapisu danych na nośnikach pamięci masowej. System operacyjny pośredniczy między użytkownikiem i programami użytkowymi a sprzętem komputerowym. Dzięki temu twórcy programów użytkowych mogą łatwo wykorzystywać możliwości sprzętu nie zagłębiając się w techniczne aspekty ich budowy i funkcjonowania. Z drugiej zaś strony producenci sprzętu komputerowego mogą w prosty sposób dostosowywać swoje produkty do pracy z różnymi systemami poprzez dostarczanie wraz z nimi sterowników urządzeń.

Jak więc widzicie - jego rola w systemie komputerowym jest równie istotna jak sprzętu. Od jego wydajności i możliwości zależy w dużej mierze komfort pracy z systemem komputerowym oraz zakres prac, które można na nim wykonać.

Komentarz

Mamy nadzieję, że teraz łatwiej Wam zrozumieć, dlaczego wśród największych firm technologicznych świata bardzo często znajdują się firmy budujące platformy, oprogramowanie i ekosystemy usług (np. Microsoft, Apple, Alphabet, NVIDIA) — a nie tylko producenci sprzętu. Kolejność w takim rankingu zmienia się w czasie, ale kierunek trendu pozostaje podobny. (Źródło: The Motley Fool: Largest companies by market cap; oraz CompaniesMarketCap: Largest tech companies by market cap)

Systemy operacyjne przeszły wiele zmian w czasie swego istnienia, ich ewolucja jednak się nie zakończyła. Współcześnie można zauważyć wydzielanie się dwóch wyraźnych nurtów w ich rozwoju:

- istnieje grupa systemów uniwersalnych - takich jak Windows, macOS czy Linux - w przypadku których dalszy rozwój jest skoncentrowany na zapewnieniu ... jeszcze większej uniwersalności. Wymienione systemy można stosować jako system operacyjny dla komputerów osobistych równie dobrze jak system stanowiący podstawę dużego serwera internetowego czy nawet (głównie Linux) jako podstawowy system operacyjny superkomputera. I to wszystko przy wykorzystaniu tego samego jądra, tej samej powłoki i dużej części wspólnych programów narzędziowych. Właściwości najnowszych wersji wspomnianych systemów przedstawimy w jednej z następnych lekcji;
- mimo ekspansji systemów uniwersalnych istnieje także duża grupa zastosowań, gdzie do tej pory się nie sprawdziły. Są to rozwiązania o specyficznych wymaganiach i / lub ograniczeniach sprzętowych. Tak więc powstały i dynamicznie rozwijają się specjalizowane systemy operacyjne, dostosowane do jednego i tylko jednego zadania. Najpopularniejszą grupą takich systemów (z którą większość z Was już się zetknęła) są systemy dla smartfonów i tabletów, takie jak Android i iOS (dominujące na rynku). Obok nich istnieją także rozwiązania niszowe lub regionalne, np. KaiOS czy HarmonyOS StatCounter: Mobile OS market share worldwide. Inna szeroka gama zastosowań to systemy czasu rzeczywistego, takie jak QNX czy VxWorks, szeroko wykorzystywane w sterownikach i systemach wbudowanych, m.in. w automatyce przemysłowej oraz systemach motoryzacyjnych. (Źródło: QNX: Automotive / Driver Assistance) W tym podręczniku potraktowaliśmy je nieco po macoszemu, większość uwagi poświęcając systemom uniwersalnym - lecz nie można o nich zapominać;

Czy te dwa nurty utrzymają się - czas pokaże. Jedyne co możemy na pewno powiedzieć to to, że możemy spodziewać się dalszego rozwoju i zmian w systemach operacyjnych nowych generacji.

Spis treści

1. Pamięć, procesy, wątki
1.1. Procesy
1.2. Podział czasu procesora
1.3. Wątki
1.4. Synchronizacja

1. Pamięć, procesy, wątki

W tej części zajmiemy się nieco dokładniej tym, jak działa jądro systemu operacyjnego, jakie mechanizmy musi implementować oraz jakie algorytmy do tego wykorzystuje.

Zaczniemy od zarządzania pamięcią operacyjną. System jako taki powinien umożliwić jej ochronę (to przede wszystkim), dlatego pokażemy Wam jak to się robi. Sama ochrona to często za mało. Powiemy więc o mechanizmach umożliwiających tworzenie tzw. pamięci wirtualnej, o pamięci swap i mechanizmie nakładek.

W dalszej części zajmiemy się procesami. Przedstawimy definicję pojęć (zarówno procesu, jak i wątku) a następnie postaramy się pokazać, w jaki sposób możliwe jest działanie wielu programów na jednym systemie komputerowym w tym samym czasie. I to działania bezpiecznego - kiedy to programy mogą w tym samym czasie korzystać ze współdzielonych zasobów.

Nie będzie to prosta lekcja, będzie jedną z trudniejszych ... na pocieszenie powiemy, że potem to już będzie tylko łatwiej 😊

Pamięć - rola, adresowanie

Rola pamięci operacyjnej

Pamięć operacyjna jest jedną z podstawowych części systemu komputerowego. Do niej trafiają niemal wszystkie dane programów (a także i same programy - zostanie to wyjaśnione później), które będą przetwarzane przez procesor. To z tej właśnie pamięci, procesor doczytuje potrzebne informacje i to w tej pamięci znajduje się system operacyjny, który nadzoruje działanie całego komputera. Ale nie tylko - do pamięci operacyjnej załadowanych jest też wiele sterowników urządzeń I/O (nawet w historycznych PC z MS-DOS wykorzystywano usługi BIOS/firmware, które były przechowywane w pamięci nieulotnej, ale po starcie komputera wczytywane do RAM dla zwiększenia wydajności).

Pamięć operacyjną można traktować jako bardzo długą tablicę komórek (najczęściej bajtów), z których każda ma swój numer, czyli adres. Procesor odwołuje się do danych i rozkazów właśnie przez podawanie takich adresów. To, jak duży „kawałek" danych procesor najwygodniej przetwarza naraz (np. 32 lub 64 bity), wynika głównie z jego architektury: rejestrów i zestawu rozkazów. Natomiast szerokość ścieżek danych wpływa przede wszystkim na to, ile danych można przesłać w jednym cyklu, czyli na przepustowość.

Podczas wykonywania programu, jednostka centralna zbiera rozkazy z pamięci, zależnie od wartości licznika rozkazów. Typowy cykl wykonania rozkazu zaczyna się pobraniem rozkazu z pamięci, później następuje dekodowanie rozkazu i jeśli to konieczne z pamięci pobierane są argumenty. Po wykonaniu rozkazu wyniki jego działania mogą zostać zachowane również w pamięci. Procesor wykonuje rozkazy pobierając je z przestrzeni pamięci adresowalnej. W komputerach osobistych jest to w praktyce pamięć RAM (wspomagana przez cache), natomiast w systemach wbudowanych spotyka się także wykonywanie kodu bezpośrednio z pamięci nieulotnej mapowanej w przestrzeń adresową (tzw. execute-in-place).

Pamięć operacyjna zwykle podzielona jest na dwie części. Jedną wykorzystuje system operacyjny, drugą programy użytkownika. W pierwszych pseudo-systemach operacyjnych, zdolnych do uruchomienia tylko jednego procesu cała pamięć operacyjna zamontowana w systemie była dostępna dla tego procesu. Z momentem pojawienia się systemów operacyjnych pojawiło się zagadnienie dzielenia pamięci na obszar systemu i programu. Istnieją różne rozwiązania (OS w pamięci ROM, pamięć RAM przeznaczona dla programu, OS w pamięci RAM, na początku przestrzeni adresowej, część OS w RAM, sterowniki urządzeń w ROM). W systemach wbudowanych spotyka się przechowywanie systemu w pamięci nieulotnej (najczęściej flash: NOR/NAND, eMMC/UFS), a do RAM trafiają dane i te fragmenty kodu, które muszą być modyfikowalne. W części urządzeń możliwe jest też wykonywanie kodu bezpośrednio z pamięci mapowanej (XIP).

Jak wspomniano, na początku, w systemach jednoprocesowych nie istniał problem przydziału i chronienia fragmentów pamięci przed różnymi procesami. W momencie powstania pierwszych systemów wieloprogramowych, np. OS/360 zagadnienie to zaczęło być analizowane a pierwsze, dość proste rozwiązania optymalizowane. W systemach wieloprogramowych pamięć przeznaczona na programy użytkownika musi być dalej podzielona tak, aby mogła współpracować z wieloma procesami. Zadaniem tym zajmuje się właśnie system operacyjny i nosi ono nazwę **zarządzania pamięcią**.

Adresowanie pamięci

W prostym modelu przydziału pamięci (bez stronicowania) system operacyjny umieszcza proces w jednym ciągłym obszarze pamięci fizycznej, a sprzęt (np. rejestr bazowy i limit) może pilnować, by proces nie wychodził poza swój przydział. W takim modelu niedopuszczalna jest sytuacja jak na poniższym rysunku, kiedy obszar procesu A byłby „pocięty" przez blok pamięci procesu B. We współczesnych systemach z MMU i stronicowaniem proces widzi ciągłą przestrzeń adresową logicznie/wirtualnie, ale fizycznie strony mogą znajdować się w nieciągłych ramkach — i to jest normalne.



Niedopuszczalny sposób alokacji pamięci w modelu bez stronicowania (wymagającym ciągłego obszaru)

Niedopuszczalność takiej sytuacji wynika stąd, że dowolny program komputerowy w postaci skompilowanej i zlinkowanej (czyli języka maszynowego) w tym modelu jest umieszczany w jednej przestrzeni adresowej, która nie może być podzielona. Dodatkowo, program operuje adresami logicznymi, a w rzeczywistości system operacyjny umieszcza go w pewnym miejscu pamięci fizycznej. Z tego powodu SO musi zapewnić relokację (tłumaczenie) adresów pamięci, z których korzysta uruchomiony proces. Może się to odbywać na dwa sposoby – poprzez modyfikację kodu maszynowego procesu (rozwiązanie obecnie niewykorzystywane) lub poprzez modyfikację adresów w momencie odwoływania się do nich przez proces (np. poprzez dodawanie do oryginalnego adresu wartości specjalnego rejestru sprzętowego przechowującego informację o początku przestrzeni adresowej aktualnego procesu). Dodanie kolejnego rejestru, określającego koniec przestrzeni adresowej procesu pozwala na sprzętowe zabezpieczenie nadpisywania pamięci należącej do innych procesów przez niewłaściwie napisany program. We współczesnych systemach ogólnego przeznaczenia rolę tę realizuje przede wszystkim MMU: sprzęt tłumaczy adresy wirtualne na fizyczne na podstawie tablic stron (a także buforów typu TLB). (Źródło: ARM: The Memory Management Unit; oraz Intel SDM)

Dlatego też mamy do czynienia z dwoma typami adresów:

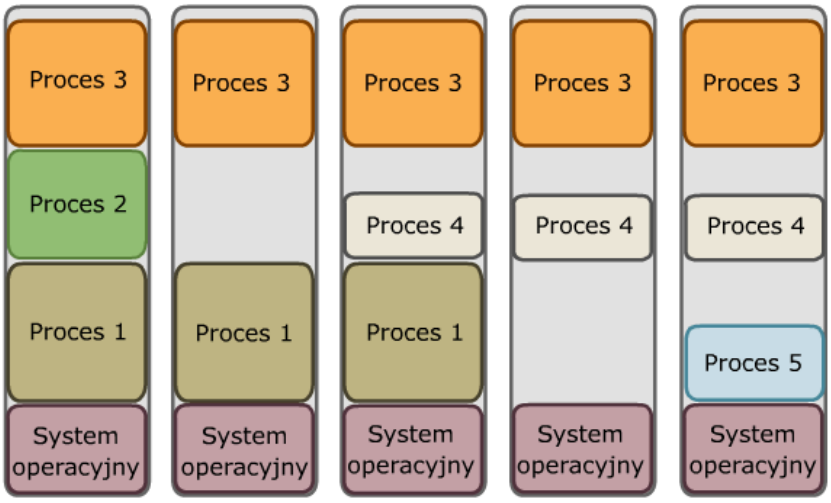
- **adres logiczny** definiowany jest jako położenie względem początku programu;
- **adres fizyczny** czyli położenie w pamięci głównej o unikatowym numerze.

Partycjonowanie

Terminem **partycjonowanie** określa się podział pamięci w celu udostępnienia jej wielu procesom. Partycjonowaniu podlega ta część pamięci, której nie zajmuje system operacyjny. Wyróżniamy dwa rodzaje partycjonowania, którym odpowiadają dwa rodzaje partycji:

1. Partycje o **ustalonym rozmiarze** - wprowadzany do pamięci proces umieszczany jest w najmniejszej partycji, w której jest w stanie się pomieścić. Wyjaśnijmy, iż termin "ustalona" nie oznacza, że rozmiary partycji są równe, wręcz przeciwnie - nie są. Takie rozwiązanie sprawi jednak, że pojemność pamięci będzie niecałkowicie wykorzystana, ponieważ mało prawdopodobne (ale możliwe) jest zdarzenie, w którym rozmiar procesu będzie równy rozmiarowi przydzielonej mu partycji.
2. Partycje o **zmiennych rozmiarach** - proces wprowadzany do pamięci jest umieszczany w partycji o dokładnie takiej pojemności, jakiej ten proces wymaga. Metoda ta jest bardziej efektywna od opisanej powyżej.

Z partycjonowaniem wiążą się pewne problemy. Rozważmy na przykład sytuację przedstawioną na rysunku:



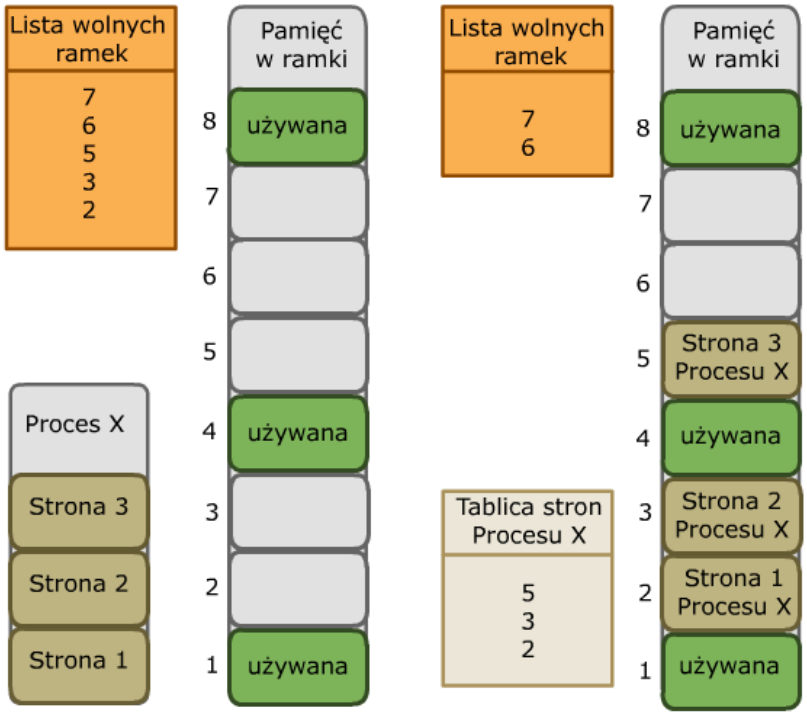
Hipotetyczny wynik partycjonowania

Na początku proces 2 zostaje usunięty z pamięci, na jego miejsce wchodzi proces 4, następnie wychodzi proces 1 i wchodzi proces 5. Na przykładzie tym doskonale widać, jak schemat partycjonowania doprowadził do powstania luk w pamięci. Zauważmy, że im dłużej trwa partycjonowanie, tym więcej powstaje takich luk i tym mniejsze procesy mogą zostać do pamięci wprowadzane. Takie postępowanie z pewnością doprowadziłoby do zapełnienia pamięci, bez jednoczesnego wykorzystania całej jej pojemności. Dlatego też stosuje się **upakowywanie** (ang. *compaction*) , czyli okresowe przesuwanie przez system operacyjny procesów w pamięci, w celu skumulowania wolnej pamięci w jednym bloku.

Stronicowanie

Wydajniejszym sposobem przydziału pamięci jest **stronicowanie**. Stronicowanie opiera się na koncepcji podziału pamięci na małe fragmenty, zwane **ramkami**, przy równoczesnym podziale pamięci procesów na takie same fragmenty, zwane **stronami**. Przy takim schemacie tracona pojemność stanowi jedynie część ostatniej strony. Jeżeli przypomnimy sobie, że strony mają małe rozmiary, zorientujemy się, że tracona pojemność również jest niewielkich rozmiarów.

Zasadę działania stron i ramek ilustruje następny rysunek, który przedstawia pewien jeszcze mało znany proces X składający się z trzech stron. Gdy proces ów wczytywany jest do pamięci, system operacyjny dysponując listą wolnych ramek (którą sam utrzymuje), potrafi w odpowiedni sposób załadować strony do wolnych, niekoniecznie sąsiadujących ramek.



Stronicowanie

Ale skąd proces ma wiedzieć, w których ramkach znajdują się jego strony? Odpowiedź jest bardzo prosta. System tworzy dla każdego procesu **tablicę stron**. Pokazuje ona gdzie są ramki pasujące do stron procesu. W przypadku stronicowania, tłumaczenie przez procesor adresów logicznych (numer strony, adres względny) umożliwia właśnie tablica stron. Procesor uzyskując do niej dostęp może, znając adres logiczny, utworzyć adres fizyczny (numer ramki, adres względny).

Stronicowanie zapewnia bardzo efektywne wykorzystanie pamięci operacyjnej przy minimalnych stratach. Ułatwiony jest również dostęp do pamięci.

Segmentacja

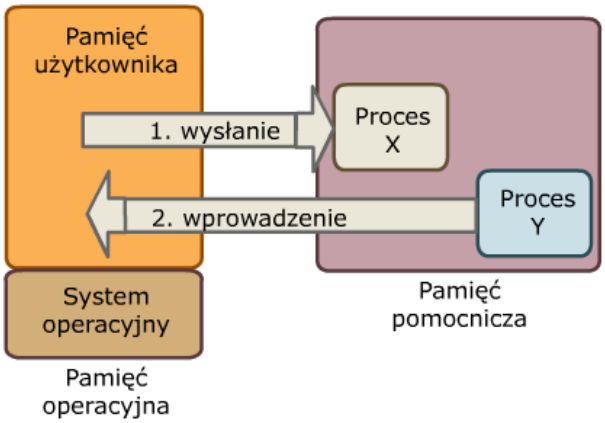
Segmentacja dzieli przestrzeń adresową procesu na kilka logicznych części, zwanych **segmentami**, które mają zmienny rozmiar. Segment może odpowiadać np. kodowi programu, danym, stosowi lub pojedynczej bibliotece. Każdy segment ma swoją bazę i rozmiar oraz może mieć przypisane prawa dostępu (np. tylko do odczytu, wykonywanie kodu) i poziom uprawnień. Odwołanie do pamięci ma wtedy postać: *(numer segmentu, adres względny)*, a sprzęt/system przelicza to na adres fizyczny.

Stronicowanie (omówione wcześniej) dzieli pamięć na małe, równe kawałki (strony i ramki), głównie po to, żeby łatwo przydzielać i przenosić pamięć bez „dziur”. **Segmentacja** ma inny cel: ułatwia porządkowanie programu i ochronę jego części, bo operuje na logicznych fragmentach o różnym rozmiarze. Wadą segmentacji jest to, że zmienne rozmiary segmentów mogą powodować podobny problem jak partycjonowanie (powstawanie luk i konieczność upakowywania). Dlatego w praktyce łączy się oba podejścia: segment wskazuje „logiczną część programu”, a wewnątrz segmentu pamięć jest przydzielana stronicami.

Swapping

Współczesne systemy operacyjne (a także sprzęt komputerowy) pozwalają na implementację zaawansowanych metod zarządzania pamięcią. Do tej grupy zalicza się **wymiana** (ang. *swapping*). Historycznie przez swapping rozumiano nawet wymianę całych obrazów procesów. We współczesnych systemach częściej chodzi o wymianę stron pamięci: system przenosi rzadko używane strony do przestrzeni swap/pagefile, a przy potrzebie wczytuje je z powrotem. (Źródło: Red Hat: Swap space; oraz Microsoft: Introduction to the page file)

Dla środowiska sterowanego rotacyjnym algorytmem planowania przydziału procesora (model historyczny) cykl wymiany wygląda następująco:



Wymiana dwóch procesów pomiędzy pamięcią operacyjną a pomocniczą.

Po wyczerpaniu przez proces X przeznaczonego mu kwantu czasu procesora, a więc tymczasowego zakończenia działania, system operacyjny wymienia go na proces Y, któremu również przydzielany jest kwant czasu procesora. Po upływie tego czasu również ten proces zostanie wymieniony.

Przy procedurach wymiany istotną rolę odgrywa pamięć pomocnicza (ang. *backing store*). Musi być ona dostatecznie pojemna, aby pomieścić kopie obrazów pamięci wszystkich użytkowników oraz umożliwić bezpośredni dostęp do tych obrazów pamięci. Wszystkie procesy gotowe do działania, których obrazy pamięci znajdują się w pamięci pomocniczej lub operacyjnej, przechowywane są przez system operacyjny w **kolejce procesów gotowych** (ang. *ready queue*). Za każdym razem, gdy system operacyjny chce wykonać proces, wywołuje ekspedytora. **Ekspedytor** (ang. *dispatcher*) sprawdza dostępność procesu (nazwijmy go X) w pamięci operacyjnej. Jeżeli proces (X) nie znajduje się w pamięci operacyjnej oraz brak jest wolnego miejsca w tejże pamięci, to *dispatcher* wysyła na dysk jeden z procesów znajdujących się w pamięci operacyjnej (np. Y). Na zwolnione w ten sposób miejsce wprowadzany jest potrzebny proces (X). Na koniec ekspedytor uaktualnia rejestry i przekazuje sterowanie do wprowadzonego procesu (X).

Wadą wymiany jest dość długi czas potrzebny na przełączenie zadania, w stosunku do krótkiego czasu na ich wykonanie. Z tego względu klasyczna technika wymiany całych procesów jest stosowana coraz rzadziej przez systemy operacyjne - a przynajmniej na tyle rzadko na ile to możliwe.

Pamięć wirtualna

Pamięć wirtualna to mechanizm, w którym każdy proces otrzymuje własną przestrzeń adresową (adresy wirtualne), a sprzęt i system operacyjny tłumaczą te adresy na adresy fizyczne. Część danych może być w razie potrzeby przechowywana w RAM, a część — czasowo — w pamięci pomocniczej (swap/pagefile) lub w plikach mapowanych do pamięci. (Źródło: ARM: MMU; Microsoft: Virtual memory functions; oraz kernel.org: Swap Management)

Pamięć wirtualna wykorzystuje schemat **stronicowania na żądanie**, które tym różni się od omówionego wcześniej *stronicowania*, że strony procesu są wprowadzane do pamięci operacyjnej dopiero wówczas, gdy jest to konieczne tj. na żądanie.

Schemat stronicowania na żądanie umożliwia wczytanie tylko tych stron procesu, których realizacja (a więc tym samym obecność w pamięci operacyjnej) jest konieczna. Pozostałe strony procesu mogą przebywać w pamięci pomocniczej. W przypadku, gdy program rozgałęzi się do rozkazu znajdującego się poza pamięcią lub nastąpi odwołanie do danych spoza obszaru pamięci, sygnalizowany jest **błąd strony**, co zmusza system do dostarczenia wymaganej strony.

Ograniczenie liczby stron procesu znajdujących się jednocześnie w pamięci operacyjnej powoduje, że w RAM można utrzymywać większą liczbę procesów. Strony, do których program się aktualnie nie odwołuje, nie muszą zajmować miejsca w pamięci operacyjnej (mogą pozostać w pamięci pomocniczej), dzięki czemu RAM jest lepiej wykorzystywany. System operacyjny musi jednak dobierać strony do usunięcia z pamięci (tzw. algorytm wymiany stron) tak, aby nie usuwać tych, które za chwilę będą znów potrzebne. W przeciwnym wypadku może dojść do **szamotania** (ang. *thrashing*), polegającego na tym, że system spędza większość czasu na obsłudze błędów stron i przeładowywaniu stron między RAM a pamięcią pomocniczą, zamiast na wykonywaniu właściwych obliczeń.

Technika stronicowania na żądanie pozwala na stworzenie procesu o rozmiarach większych niż pojemność pamięci głównej oraz dała programistom ogromne zaplecze - pamięć wirtualną.

1.1. Procesy

Definicja procesu

Do tej pory już często przewijało się pojęcie procesu. Często używaliśmy go zamiennie z bardziej intuicyjnym określeniem program ... nie zawsze jest to podejście prawidłowe. Oba pojęcia występują w informatyce i co więcej - nie są sobie równoważne. W systemach nie umożliwiających uruchomienia więcej niż jednego programu naraz nie istniał problem rozdzielania dostępu do poszczególnych urządzeń składowych komputera różnym programom, nie było też potrzeby zatrzymywania i uruchamiania kolejnych procesów. Wprowadzenie możliwości uruchomienia i wykonywania więcej niż jednego programu jednocześnie spowodowało konieczność rozwiązania przedstawionych wyżej zagadnień.

> **Definicja**
>
> Przez **proces** rozumiem uruchomiony program będący w trakcie wykonywania, wraz z danymi i zasobami, które system operacyjny mu przydzielił (np. pamięcią oraz dostępem do plików i urządzeń). Proces jest więc „aktywną" postacią programu w systemie operacyjnym.

Na pojęcie procesu, oprócz kodu programu, składają się również: bieżąca czynność reprezentowana przez **licznik rozkazów** (ang. *program counter*) oraz zawartość rejestrów procesora, zwykle także **stos procesu** (ang. *process stack*), **sekcja danych** (ang. *data section*) zawierająca zmienne globalne, oraz odwołania do zasobów przydzielonych przez system operacyjny np. uchwytów (handle) w Microsoft Windows. Często spotkacie się także z określeniem **kontekst wywołania** (execution context). Kontekst ten to zestaw danych, dzięki którym SO jest w stanie nadzorować wykonywanie programu. Do kontekstu należy zawartość rejestrów procesora (licznika programu i rejestrów danych), priorytet danego procesu oraz informacja o tym czy proces oczekuje na wystąpienie określonego zdarzenia (event). Każdy proces charakteryzuje się jednocześnie stanem procesu. Nie mylcie jednak pojęcia procesu z wątkiem opisanym dalej — w tym rozdziale skupię się na procesach, a wątki omówię osobno w następnym fragmencie podręcznika.

Ważne jest rozgraniczenie pojęcia programu i procesu. Pamiętajmy, że program jest obiektem pasywnym, a wszelka aktywność należy do procesów. Po uruchomieniu każdemu procesowi przydzielone zostają zasoby, czyli:

- czas procesora (w systemach wieloprocesorowych także możliwość wykonywania na różnych procesorach/rdzeniach);
- pamięć (czyli stos procesu i sekcja danych);
- dostęp do urządzeń wejścia-wyjścia;
- pliki.

Stan procesu

Proces, który się wykonuje, zmienia również swój stan (ang. *state*). Stan ten jest częściowo określany przez to, co robi proces w chwili bieżącej. Możliwe są następujące stany procesów:

- **nowy** - pierwszy stan procesu;
- **aktywny** - proces wykonuje instrukcje;
- **oczekiwanie** - proces czeka na wystąpienie jakiegoś zdarzenia (np. zakończenie operacji wejścia-wyjścia);
- **gotowy** - proces czeka na przydział procesora;
- **zakończony** - proces zakończył działanie.

W różnych systemach operacyjnych stany noszą różne nazwy. W niektórych omówione stany są bardziej rozdrobnione, jednak przedstawione powyżej stany występują zawsze. Pamiętajmy, że w każdej chwili, w określonym procesorze, tylko jeden proces może być aktywny, zaś wiele procesów może być gotowych lub oczekujących.

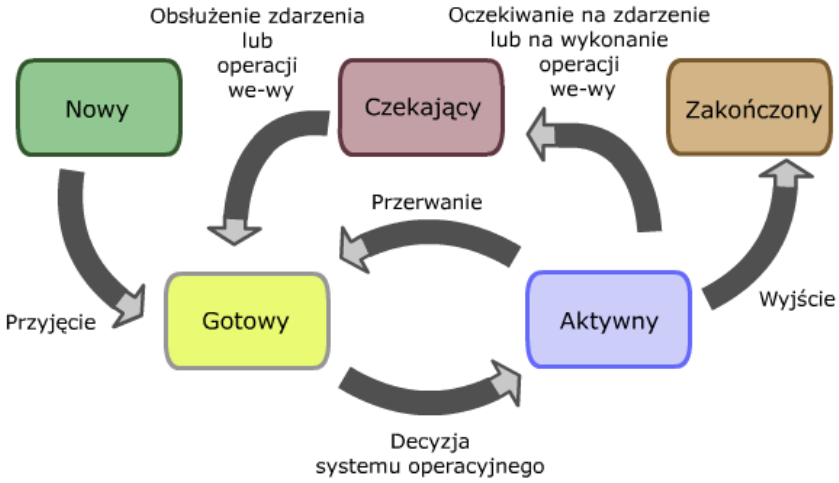


Diagram stanów procesu.

Powyższy rysunek przedstawia wszelkie możliwe przejścia pomiędzy poszczególnymi stanami procesu.

Blok kontrolny procesu

System operacyjny przechowuje informacje o uruchomionych procesach w specjalnym miejscu pamięci operacyjnej zwanym **tablicą procesów** (process table). Każdy proces reprezentowany jest tam przez **blok kontrolny procesu** (ang. *process control block* - PCB). Blok kontrolny procesu zawiera następujące pola:

- **stan procesu** - jeden ze stanów procesu;
- **licznik rozkazów** - adres następnego rozkazu do wykonania w procesie;
- **rejestry procesora** - lista i typ rejestrów zależą od procesora;



Blok kontrolny procesu

- **informacje o planowaniu przydziału procesora** - należy do nich priorytet procesu, wskaźniki do kolejek planowania (np. kolejki procesów gotowych) i inne parametry planowania;
- **informacje o zarządzaniu pamięcią** - zawartości rejestrów granicznych, tablice stron i segmentów;
- **informacje do rozliczeń** - należą do nich: ilość zużytego czasu procesora i czasu rzeczywistego, ograniczenia czasowe, numery kont, numery zadań lub procesów.
- **informacje o stanie wejścia-wyjścia** - występują tu informacje o urządzeniach I/O przydzielonych do procesu, wykaz otwartych plików, itp.

Dzięki tym informacjom SO może w dowolnym momencie zatrzymywać i uruchamiać procesy oraz chronić dane obszary pamięci, co pozwala na współbieżne działanie wielu programów.
W danej chwili na jednym rdzeniu procesora wykonywany jest tylko jeden proces. Jeżeli uruchomionych jest więcej procesów niż rdzeni, jądro przełącza wykonywanie między nimi, przydzielając im krótkie odcinki czasu.

Tworzenie procesów

Proces może utworzyć kolejne procesy. Mówimy, że proces "stwórca" - to **proces macierzysty** (ang. *parent process*), zaś proces przez niego utworzony - to **proces potomny** lub **potomek** (ang. *child*).

Dopuszczalne są dwie możliwości przydziału przestrzeni adresowej dla procesu potomnego:

- proces potomny jest kopią swego rodzica;
- proces potomny wiąże się z wykonaniem nowego programu.

Ponieważ proces do wykonania powierzonych mu zadań potrzebuje zasobów systemowych (np. czasu procesora), to muszą one zostać przydzielone. Proces potomny utworzony przez proces macierzysty może dostać zasoby systemowe od systemu operacyjnego lub też od procesu macierzystego - przy czym ta druga droga jest rzadko wykorzystywana. Po utworzeniu procesu potomnego, proces macierzysty może:

- kontynuować działanie współbieżnie ze swoimi potomkami;
- zaczekać na zakończenie działań swoich procesów potomnych (części z nich albo wszystkich).

Drugi z wyżej wymienionych przypadków pozwala na zastosowanie w środowiskach wieloprocesorowych mechanizmów synchronizacji (np. barier).

Kończenie procesu

Koniec działania procesu następuje, gdy proces wykona swoją ostatnią instrukcję. Wówczas proces wywołuje funkcję systemową *exit* i zostaje usunięty przez system operacyjny. Przed swym wyjściem może jednak przekazać dane do procesu macierzystego. Przed wyjściem odbierane są mu również wszystkie zasoby procesu przez system operacyjny.

Zakończenie procesu może przebiegać także inaczej. Proces macierzysty może zakończyć działanie procesu potomnego korzystając z mechanizmów systemu operacyjnego. Natomiast funkcja *abort()* dotyczy zakończenia bieżącego procesu (zwykle w sposób awaryjny). (Źródło: cppreference: abort)

Do zakończenia procesu potomnego proces macierzysty musi znać identyfikatory swych potomków - więc tworzenie nowego procesu zawsze wiąże się z przekazaniem jego identyfikatora do procesu macierzystego. Zakończenie procesu potomnego może zostać użyte w przypadku, gdy:

- wykonanie zadania przez potomka jest już zbędne;
- potomek nadużył przydzielonych mu zasobów;
- po zakończeniu procesu macierzystego system operacyjny nie pozwala jego potomkowi na dalszą pracę.

W przypadku gdy proces macierzysty nie obsłuży odpowiednio zakończenia procesu potomnego (w uproszczeniu - nie odbierze informacji wygenerowanej przez system o zakończeniu procesu potomnego) - w tablicy procesów pozostaje wpis opisujący program, którego wykonanie w systemie operacyjnym zostało zakończone. Taki proces nazywa się **procesem zombie**. Termin ten odnosi się zazwyczaj do systemów z rodziny Unix, gdzie pozostawienie wpisu zombie tymczasowo zajmującego pozycję w tablicy procesów zapobiega ponownemu wykorzystaniu identyfikatora procesu i możliwym na skutek tego pomyłkom programistycznym.

Współpraca pomiędzy procesami

We współczesnym systemie operacyjnym współbieżnym procesy, które są w trakcie wykonania, mogą współpracować ze sobą lub działać niezależnie. W praktyce domyślnie procesy są od siebie odseparowane (mają własne przestrzenie adresowe), a współpraca wymaga użycia kontrolowanych mechanizmów komunikacji międzyprocesowej (IPC). (Źródło: Microsoft: Virtual Address Spaces; oraz Microsoft: Interprocess Communications)

Definicja

Proces niezależny (ang. *independent*) to taki, który nie wymienia danych z innymi procesami w trakcie działania i nie korzysta ze wspólnych obszarów pamięci ani z mechanizmów IPC. Procesy są więc niezależne wtedy, gdy wynik ich działania nie zależy od innych procesów (np. prosty program obliczeniowy uruchomiony „dla siebie”, który tylko czyta dane wejściowe i zapisuje wynik do pliku).

Definicja

Proces współpracujący (ang. *cooperating*) to taki, który celowo komunikuje się z innymi procesami lub współdzieli z nimi pewne dane. Taka współpraca jest realizowana przez mechanizmy IPC, np. potoki (pipe / named pipe), gniazda (socket), kolejki komunikatów, pamięć współdzieloną, sygnały lub pliki tymczasowe. (Źródło: Microsoft: IPC (Win32); Microsoft: Pipes; man7: shm_open(); Oracle: POSIX IPC)

Dzięki wprowadzeniu środowiska umożliwiającego współpracę pomiędzy procesami możliwe staje się:

- dzielenie informacji - np. jeden proces zbiera dane (z sieci, z czujników, z plików), a inny je prezentuje lub przetwarza; klasyczny przykład to aplikacja kliencka komunikująca się z serwerem (lokalnie lub po sieci);
- przyspieszenie obliczeń - duże zadania mogą być dekomponowane na podzadania wykonywane równolegle przez osobne procesy (np. renderowanie, kodowanie wideo, analiza dużych danych), a następnie scalane w wynik końcowy;
- modularność - system można budować z wielu mniejszych procesów pełniących różne role (np. osobny proces dla interfejsu użytkownika, osobny dla obliczeń w tle, osobny dla komunikacji z urządzeniem); w razie awarii jednego elementu pozostałe mogą nadal działać;
- wygoda - możliwość korzystania z kilku funkcji systemu równocześnie (np. redagowanie tekstu i drukowanie), gdzie wydruk realizuje oddzielny proces/usługa, a aplikacja użytkownika nie musi „czekać” na zakończenie całej operacji.

Zagadnienia współpracy pomiędzy procesami, oraz pomiędzy omówionymi dalej wątkami są jedną z największych zmór programistów - często doprowadzając do sytuacji, w której program (programy) nie działają i nikt nie wie dlaczego ... ;) Trochę dalej w tym podręczniku postaram się powiedzieć Wam dlaczego ...

1.2. Podział czasu procesora

W systemie, w którym uruchomione jest więcej procesów niż rdzeni procesora, jądro systemu operacyjnego wykonuje je na przemian, uruchamiając każdy z procesów na krótki okres czasu. Czasy przełączania oraz wykonywania poszczególnych fragmentów procesów są na tyle krótkie, że dla użytkownika systemu wykonywanie procesów wydaje się być jednoczesne. Opisany mechanizm przemiennego wykonywania programów realizowany jest przez systemy operacyjne z tzw. **wywłaszczającą wielozadaniowością** (ang. *preemptive multitasking*). W tym trybie działają współczesne systemy ogólnego przeznaczenia (Microsoft Windows, Linux, macOS) oraz nowoczesne systemy mobilne (Android, iOS). (Źródło: Microsoft: Multitasking (Win32))

Historycznie spotykano także **wielozadaniowość współpracującą** (ang. *cooperative multitasking*), np. w klasycznych wersjach Windows 1.x–3.x oraz w klasycznym Mac OS (do serii 9). Różnica w działaniu polega na tym, że w przypadku uruchomienia wielu programów pierwszy z nich jest wykonywany ciągle do momentu, aż sam odda sterowanie (np. przejdzie w stan oczekiwania na zdarzenie). W takim podejściu jedna aplikacja, która „nie oddaje sterowania", potrafiła pogorszyć responsywność całego środowiska (wyobraźcie sobie co działoby się, gdyby takie zasady zastosować w polskim Sejmie - jak ktoś wejdzie na mównicę - to może być na niej dowolnie długo, aż do momentu jak sam stwierdzi że odda głos następnemu ...). W praktyce to rozwiązanie nie sprawdziło się najlepiej w systemach ogólnego przeznaczenia i współcześnie nie jest stosowane jako podstawowy mechanizm planowania w jądrze nowoczesnych systemów desktopowych i serwerowych.

Wracając do systemów z wywłaszczającą wielozadaniowością - by to rozwiązanie działało optymalnie, należy zapewnić algorytmy **planowania przydziału czasu** procesora, czyli określenia kolejności przydziału jego mocy obliczeniowej zadaniom z *kolejki procesów gotowych do działania.* Jeszcze prościej - określenie który z procesów ma być właśnie uruchomiony i na jak długo. (Do tego, co w praktyce jest bezpośrednio planowane przez jądro, wrócę w rozdziale o wątkach.)

Część systemu operacyjnego odpowiedzialna za tę funkcjonalność nazywana jest **schedulerem** (polski odpowiednik - planista - raczej się nie przyjął). W zależności od przeznaczenia systemu operacyjnego (np. dla komputera osobistego, smartfona, serwera) algorytmy sterujące wykonaniem procesu mogą być różne, bardziej lub mniej skomplikowane. Generalnie scheduler ma kilka zadań:

- wybranie odpowiedniego procesu do uruchomienia (w większości systemów operacyjnych procesy mogą mieć różne priorytety, oznaczające ich istotność w systemie i warunkujące w pewnym stopniu kolejność ich uruchamiania);
- zapewnienie jak najlepszego wykorzystania czasu procesora;
- minimalizację czasu jaki procesy oczekują na uruchomienie;
- minimalizację liczby procesów oczekujących (czyli możliwie szybkie zakończenie procesów uruchomionych);
- przydział takiego samego czasu procesora każdemu procesowi (z uwzględnieniem jego priorytetu);
- w szczególnych przypadkach (systemy czasu rzeczywistego) zapewnienie wykonania procesu przed upłynięciem wyznaczonego czasu;
- maksymalizację wykorzystania urządzeń peryferyjnych

Dodatkowo, ze względu na mnogość operacji wykonywanych przy przełączaniu procesów (np. wejście do trybu jądra, zapisanie bieżącego stanu, przełączenie mapowania pamięci, wczytanie danych dla nowo uruchamianego procesu itd.), jest to operacja kosztowna czasowo. Z tego względu powinna być wykonywana możliwie rzadko, co z kolei często stoi w konflikcie z wymaganiem dużej responsywności systemu.

Zanim przejdę do algorytmów, muszę wprowadzić pojęcie **fazy procesora** (czasem spotkacie też określenie *CPU burst*). W praktyce proces nie wykorzystuje procesora przez cały czas swojego życia: na przemian wykonuje obliczenia na CPU oraz czeka na zdarzenia wejścia/wyjścia (np. dysk, sieć, klawiatura) lub na dane od innych elementów systemu.

> ## Definicja
>
> Przez **fazę procesora** rozumiem ciągły odcinek czasu, w którym proces faktycznie wykonuje instrukcje na CPU, aż do momentu, gdy zostanie zablokowany (np. przez operację wejścia/wyjścia) albo zostanie mu odebrany procesor w wyniku wywłaszczenia. Algorytmy takie jak FCFS, SJF czy RR porównują i planują właśnie te odcinki czasu (fazę obliczeń), a nie „całe życie" procesu.

Poniżej zostaną omówione najciekawsze z algorytmów planowania przydziału procesora.

Algorytm FCFS

FCFS (ang. *first-come, first-served*) oznacza "pierwszy zgłoszony - pierwszy obsłużony". Algorytm ten opiera się na prostej zasadzie, którą właściwie wyjaśnia jego nazwa. Najlepiej do implementacji algorytmu nadaje się kolejka FIFO, w której blok kontrolny procesu wchodzącego do kolejki dołączany jest na jej koniec, zaś wolny procesor przydzielany jest procesowi z początku kolejki.

Wadą wspomnianej metody przydziału procesora jest to, że średni czas oczekiwania może być bardzo długi. Jeżeli rozważymy przyjście w danym momencie czasu trzech procesów, których długości faz procesora wynoszą:

Proces	Czas trwania fazy

P_1	21 ms
P_2	6 ms
P_3	3 ms

przekonamy się, że nadejście procesów w kolejności P_1,P_2,P_3 wyglądać będzie na *diagramie Gantta* następująco dla algorytmu FCFS:



Dla procesu P_1 czas oczekiwania wynosi więc 0 ms, dla procesu P_2 21 ms, dla procesu P_3 zaś 27 ms. Średni czas oczekiwania wynosi więc: (0+21+27)/3 = 16 ms. Dla sytuacji, w której procesy nadeszłyby w kolejności P_3, P_2, P_1 diagram Gantta wyglądałby następująco:



Dla takiego przypadku średni czas oczekiwania wyniósłby: (0+3+9)/3 = 4 ms. Widzimy więc, że zastosowanie algorytmu niesie ze sobą pewne ryzyko.

W warunkach dynamicznych działanie algorytmu również nie jest zadowalające. W przypadku, gdy mamy do czynienia z dużym procesem ograniczonym przez procesor oraz małymi (szybciej się wykonującymi) procesami ograniczonymi przez wejście-wyjście, wtedy proces ograniczony przez procesor jako pierwszy uzyska dostęp do procesora, procesy wejścia-wyjścia oczekiwać będą w kolejce procesów gotowych stosunkowo długo, a urządzenia wejścia wyjścia w tym czasie nie będą mogły być używane. Po zakończeniu wykonania procesu ograniczonego przez procesor, proces ten przejdzie do kolejki oczekującej na wejście-wyjście, natomiast małe procesy ograniczone przez wejście-wyjście wykonają się w krótkim czasie i zwolnią procesor, który może wtedy być bezczynny. Następnie powróci nasz duży proces, który ponownie będzie obsługiwany przez procesor, a procesy mniejszych rozmiarów będą musiały czekać. Sytuację wyżej opisaną, w której procesy oczekują na zwolnienie procesora przez jeden duży proces, nazywamy **efektem konwoju** (ang. *convoy effect*).

Algorytm ten, z uwagi na to, że pozwala procesowi na zajmowanie procesora przez dłuższy czas, nie nadaje się do stosowania w środowiskach z podziałem czasu, w których każdy użytkownik powinien dostawać przydział procesora w równych odstępach czasu.

Komentarz

W systemach historycznych, które nie miały wywłaszczania, problemem nie był tyle konkretny algorytm planowania, co sam fakt, że aplikacja mogła zbyt długo nie oddawać sterowania. Dopiero planowanie wywłaszczające umożliwiło przewidywalną responsywność środowiska wielozadaniowego. (Źródło: Microsoft: Multitasking (Win32))

Algorytm SJF

Algorytm SJF (ang. *shortest-job-first*), czyli "najkrótsze zadanie najpierw", opiera swe działanie na powiązaniu procesu z długością jego najbliższej fazy. Procesor zostaje przydzielony procesowi, który ma najkrótszą następną fazę procesora. W przypadku, gdy dwa procesy mają następne fazy procesora równe, stosuje się algorytm FCFS.

Porównajmy działanie tego algorytmu do algorytmu FCFS. Załóżmy więc, że mamy 3 następujące procesy:

Proces	Czas trwania fazy
P_1	6 ms
P_2	9 ms
P_3	3 ms

Diagram Gantta dla tych procesów wygląda następująco:



Jeżeli procesy przychodzą w kolejności P_1, P_2, P_3 to dla algorytmu SJF średni czas oczekiwania wynosi (3+9+0)/3 = 4ms, zaś dla algorytmu FCFS wyniósłby (0+6+15)/3 = 7 ms.

Algorytm SJF daje minimalny średni czas dla danego zbioru procesów. Mówimy, że algorytm SJF jest **czaso-optymalny**.

Dużym problemem w algorytmie SJF jest określenie długości następnego zamówienia na przydział procesora. Przy planowaniu długoterminowym zadań, za czynnik ten możemy przyjąć limit czasu procesu określony przez przedkładającego zadanie użytkownika. Algorytm SJF jest więc chętnie używany w planowaniu długoterminowym. Natomiast w przypadku planowania krótkoterminowego, nie ma sposobu na określenie długości następnej fazy procesora. Dlatego też oszacowuje się tę wartość, obliczając ją na ogół jako średnią wykładniczą pomiarów długości poprzednich faz procesora.

Możliwe są dwa sposoby realizacji algorytmu SJF - wywłaszczająca (ang. *shortest-remaining-time-first*) oraz niewywłaszczająca. Różnica pomiędzy tymi realizacjami widoczna jest w sytuacji, gdy w *kolejce procesów gotowych* pojawia się nowy proces o krótszej następnej fazie procesora niż pozostała faza "resztki" procesu aktualnie wykonywanego. W takim układzie algorytm wywłaszczający usuwa aktualny proces z procesora, zaś algorytm niewywłaszczający pozwala procesowi bieżącemu na zakończenie fazy procesora. Forma wywłaszczająca algorytmu SJF jest więc szybsza.

Algorytm planowania rotacyjnego - RR

Algorytm planowania rotacyjnego (ang. *round-robin* - RR) jest oparty na algorytmie FCFS, jednakże różni się od FCFS faktem, że możliwe jest wywłaszczanie. Algorytm ten jest chętnie stosowany w systemach z podziałem czasu.

Zasada działania algorytmu jest następująca. Z każdym procesem powiązany jest kwant czasu, najczęściej 10 do 100 ms. Procesom znajdującym się w kolejce (traktowanej jako cykliczna) procesów gotowych do wykonania, przydzielane są odpowiednie odcinki czasu, nie dłuższe niż jeden kwant czasu.

Do implementacji algorytmu, podobnie jak w przypadku FCFS, używa się kolejki FIFO - brany jest pierwszy proces z wyjścia kolejki, ustawiany jest timer na przerwanie po upływie 1 kwantu czasu i proces jest wysyłany do procesora. Jeżeli czas trwania fazy procesu jest krótszy od 1 kwantu czasu, proces sam zwolni procesor i rozpocznie się obsługiwanie kolejnego procesu z kolejki. Jeżeli proces ma dłuższą fazę od 1 kwantu czasu, to nastąpi przerwanie zegarowe systemu operacyjnego, nastąpi przełączenie kontekstu, a proces do niedawna wykonywany, trafi na koniec kolejki procesów gotowych do wykonania.

Zobaczmy, ile wynosi średni czas oczekiwania w porównaniu z metodą FCFS dla następujących danych:

Proces	Czas trwania fazy
P_1	21 ms
P_2	6 ms
P_3	3 ms

Dla kroku 7 ms diagram Gantta dla powyższych procesów wyglądał będzie następująco:



Średni czas oczekiwania wynosi więc: (0+13+16)/3 = 9,67 ms.

Wydajność algorytmu rotacyjnego zależy od kwantu czasu. W przypadku, gdy kwant czasu jest bardzo długi (dąży do nieskończoności) metoda RR działa jak FCFS. Natomiast w przypadku, gdy kwant czasu jest bardzo krótki (dąży do zera) planowanie RR nazywamy **dzieleniem procesora** (ang. *processor sharing*), gdyż zachodzi złudzenie, że każdy z n procesów ma własny procesor działający z prędkością 1/n prędkości prawdziwego procesora.

W przypadku algorytmu RR, podczas mówienia o wydajności, należy zastanowić się nad zagadnieniem przełączania kontekstu. Im mniejszy jest kwant czasu, tym system bardziej jest narażony na częste przełączenia kontekstu. Jeżeli czas przełączania kontekstu wynosi 10 procent kwantu czasu, to niemalże 10 procent czasu procesora jest tracone w wyniku przełączania kontekstu.

Algorytm planowania priorytetowego

Większość użytkowników komputerów osobistych zauważyła z pewnością że wskazane jest aby niektóre zadania wykonywane były szybciej niż inne, które mogą być przesunięte w czasie. Prostym przykładem jest reakcja komputera na ruch myszy czy naciśnięcie klawisza – jeżeli system reaguje na te zdarzenia z widocznym opóźnieniem to komfort pracy jest znacznie obniżony, a użytkownik narzeka na „wolny komputer”. Jednocześnie to czy np. defragmentacja dysku lub indeksowanie jego zawartości działające w tle wykona się wolniej czy szybciej jest z punktu widzenia użytkownika nieistotne. Wynika z tego wniosek, że każde zadanie wykonywane przez komputer powinno mieć swój priorytet aby uszeregować je w zbiorze innych, równolegle wykonywanych. Aby wykorzystać informację o priorytecie procesu podstawowy algorytm planowania rotacyjnego musi zostać nieco zmodyfikowany. W tym celu stosuje się organizację uruchomionych procesów w szereg kolejek, po jednej dla każdego priorytetu.

W ramach każdej kolejki szeregowanie zadań odbywa się według zasad zwykłego algorytmu round robin, natomiast numer kolejki do uruchomienia dobierany jest tak aby faworyzowane były te z wyższym priorytetem (podobnie jak to ma miejsce w przypadku algorytmu SJF, tyle że tam kryterium był czas wykonania - tu jest priorytet kolejki).

Z takim podejściem wiąże się zagrożenie wstrzymywania zadań niskopriorytetowych w nieskończoność, nazywane **głodzeniem** (starving). W sytuacji, w której system jest bardzo obciążony, może zdarzyć się, że proces o niskim priorytecie będzie czekał w kolejce procesów gotowych w nieskończoność. Z tego powodu oryginalny algorytm planowania jest modyfikowany o dynamiczną regulację priorytetów procesów, często bazującą na ilości odwołań do systemów wejścia / wyjścia w danym procesie lub czasie oczekiwania na uruchomienie (priorytet długo oczekujących procesów jest zwiększany - fachowo nazywa się to **postarzaniem procesów**).

Algorytm FIFO dla systemów czasu rzeczywistego (SCHED_FIFO)

W systemach czasu rzeczywistego (szczególnie typu *hard*) często stosuje się proste i przewidywalne planowanie stałopriorytetowe. Jednym z najbardziej klasycznych rozwiązań jest polityka **SCHED_FIFO** (ang. *First-In, First-Out*) znana z systemów zgodnych z POSIX oraz z wielu RTOS-ów. (Źródło: Linux man-pages: sched(7); Red Hat: Scheduling policies for Real Time)

Idea jest bardzo prosta: każdy proces (zadanie) ma **priorytet**. Jeśli proces o wyższym priorytecie stanie się gotowy do wykonania, to natychmiast **wywłaszcza** proces o niższym priorytecie. Natomiast w obrębie **tego samego** priorytetu obowiązuje zasada FIFO: proces, który pierwszy wszedł do kolejki gotowych, ten pierwszy dostaje procesor. Co ważne, w SCHED_FIFO proces zwykle **nie jest** „odrywany" od procesora tylko dlatego, że minął kwant czasu – będzie wykonywany tak długo, aż sam się zablokuje (np. na I/O), dobrowolnie odda sterowanie lub zostanie wywłaszczony przez proces o wyższym priorytecie. (Źródło: QNX: FIFO scheduling (SCHED_FIFO); Linux man-pages: sched(7))

Dlaczego to jest atrakcyjne w systemach czasu rzeczywistego? Ponieważ zachowanie planisty jest łatwe do przewidzenia: jeśli krytyczny proces ma najwyższy priorytet, to w normalnych warunkach nic „mniej ważnego" nie zabierze mu czasu. Z drugiej strony trzeba pamiętać, że źle ustawione priorytety mogą doprowadzić do **głodzenia** zadań niskopriorytetowych (mogą czekać bardzo długo, jeśli zadania wysokopriorytetowe ciągle mają pracę do wykonania). (Źródło: Red Hat: Scheduling policies for Real Time)

Mały przykład intuicyjny: wyobraźmy sobie, że proces **A** (priorytet 80) obsługuje krytyczny czujnik i co pewien czas musi szybko przetworzyć próbkę. Proces **B** (priorytet 30) zapisuje dane na dysk, a proces **C** (priorytet 10) robi raport w tle. Gdy tylko **A** stanie się gotowy, natychmiast przejmie procesor od **B** lub **C**. Jeśli jednocześnie gotowe są dwa procesy o tym samym priorytecie, to w SCHED_FIFO jako pierwszy uruchomi się ten, który wcześniej trafił do kolejki.

W praktyce spotyka się też wariant podobny do FIFO, ale z „krojonym" czasem w obrębie tego samego priorytetu (SCHED_RR). W systemach czasu rzeczywistego wybór między SCHED_FIFO i SCHED_RR zależy od tego, czy ważniejsza jest maksymalna przewidywalność (FIFO), czy bardziej równy podział czasu między zadaniami o tym samym priorytecie (RR). (Źródło: Linux man-pages: sched(7); QNX: Scheduling policies)

Algorytm multi-level feedback queues (MFQ)

Rozwinięciem idei priority schedulingu jest algorytm multi-level feedback queues. Podobnie jak w przypadku planowania priorytetowego tutaj także procesy umieszczane są w kilku listach tworzonych dla każdego poziomu priorytetów.

Każda z kolejek może mieć jednak inny model zarządzania (niekoniecznie RR, a na przykład SJF) oraz wielkość kwantu czasu procesora. Generalną zasadą jest otrzymywanie krótkich kwantów przez procesy o wysokim priorytecie. Nowy proces otrzymuje wysoki priorytet i jest uruchamiany na bardzo krótki czas. Jeżeli w tym fragmencie wykonania proces nie wykonuje odwołań blokujących (czyli CPU jest zajęty przez cały kwant), to priorytet procesu jest obniżany, zwiększany za to jest jego kwant. Jeżeli proces odwołuje się do urządzeń wejścia / wyjścia to jego priorytet jest podnoszony. W efekcie najszybciej wykonywane są krótkotrwałe zadania, a także takie, które często odnoszą się do urządzeń wejścia / wyjścia (np. wprowadzanie danych przez użytkownika).

Algorytm O(1)

System operacyjny Linux w starszych wersjach jądra (przed 2.6.23) wykorzystywał scheduler nazywany O(1). Cechą charakterystyczną (od której algorytm ten wziął nazwę) było to, że koszt wyboru następnego zadania do uruchomienia był niezależny od liczby uruchomionych zadań w systemie.

Zasada działania tego algorytmu (w uproszczeniu) opierała się na dwóch zbiorach kolejek zadań – aktywnych i wygasłych (ang. *active*, *expired*). Zadanie otrzymywało określony przydział czasu, a po jego zużyciu było przenoszone do części „expired". Gdy część „active" stawała się pusta, role obu części były zamieniane.

Z czasem O(1) został zastąpiony przez CFS, między innymi dlatego, że heurystyki „interaktywności" potrafiły dawać wyniki nie w pełni zgodne z oczekiwaniami użytkownika. (Źródło: Linux kernel docs: CFS Scheduler)

Algorytm CFS

Z powodu wyżej wymienionych wad opracowano nowy algorytm planowania zadań, zwany CFS czyli *completely fair scheduler* (całkowicie sprawiedliwy scheduler). „Sprawiedliwość" oznacza tutaj przydzielanie każdemu zadaniu porcji czasu procesora w sposób zbliżony do ideału: gdy jest n aktywnych zadań, każde dostaje w przybliżeniu 1/n mocy obliczeniowej (z uwzględnieniem priorytetów). (Źródło: Linux kernel docs: CFS Scheduler)

W CFS kluczową rolę odgrywa pojęcie **czasu wirtualnego** zadania (ang. *vruntime*) - w uproszczeniu jest to „ważony" czas wykonania, który rośnie szybciej dla zadań o niższym priorytecie, a wolniej dla zadań faworyzowanych. Scheduler wybiera do uruchomienia to zadanie, które ma najmniejszy *vruntime*, czyli jest najbardziej „pokrzywdzone" w dotychczasowym przydziale czasu procesora. Do efektywnego wyszukiwania takiego zadania wykorzystywana jest struktura drzew czerwono-czarnych. (Źródło: Linux kernel docs: CFS Scheduler)

Interesującą właściwością CFS jest to, że czas, na który procesor przydzielany jest poszczególnemu zadaniu, wynika z liczby aktualnie uruchomionych zadań i jest dobierany dynamicznie. (Źródło: Linux kernel docs: CFS Scheduler)

W nowszych jądrach Linux (od wersji 6.6) klasa „normalna” schedulera jest rozwijana w kierunku algorytmu **EEVDF** (*Earliest Eligible Virtual Deadline First*). Jest to nadal planowanie proporcjonalne (sprawiedliwe), ale oparte o pojęcie „wirtualnego terminu” (ang. *virtual deadline*) i regułę wyboru zadania o najwcześniejszym takim terminie, o ile zadanie jest „uprawnione” do wykonania (eligible). (Źródło: Linux kernel docs: EEVDF Scheduler)

Intuicyjnie można to rozumieć tak: zamiast patrzeć wyłącznie na to, kto ma najmniejszy „dług” w sensie *vruntime*, EEVDF wyznacza każdemu zadaniu wirtualny termin wykonania wynikający z jego udziału w CPU oraz priorytetu, a następnie dąży do dotrzymania tych terminów. Dzięki temu łatwiej utrzymać dobrą responsywność dla zadań „wrażliwych na opóźnienia”, jednocześnie nie łamiąc zasady sprawiedliwego podziału czasu procesora. (Źródło: Linux kernel docs: EEVDF Scheduler)

Warto też zauważyć, że EEVDF nie jest „systemem czasu rzeczywistego” w sensie twardym - nadal mówimy o klasie zadań ogólnego przeznaczenia, której celem jest kompromis między sprawiedliwością, przepustowością i opóźnieniami widzianymi przez użytkownika. EEVDF jest więc odpowiedzią na praktyczny problem: jak uzyskać bardziej przewidywalną interaktywność bez rozbudowanych heurystyk. (Źródło: Linux kernel docs: EEVDF Scheduler)

Nowoczesne systemy operacyjne używają różnych technik zarządzania procesami. Dla przykładu: Windows stosuje planowanie wywłaszczające i przydziela kwant czasu wykonywanym jednostkom wykonania, przełączając je po jego upływie; Linux historycznie korzystał z CFS i rozwija ten kierunek w stronę EEVDF; a w macOS jądro XNU wykorzystuje mechanizmy wywodzące się m.in. z Mach (w tym planowanie oparte o kolejki uruchomieniowe i priorytety). (Źródło: Microsoft: Multitasking (Win32); Linux kernel docs: CFS Scheduler; Linux kernel docs: EEVDF Scheduler; Apple: Mach Scheduling and Thread Interfaces)

1.3. Wątki

Wątek (ang. *thread*) to podstawowa jednostka wykonania, którą system przydziela do procesora (wątek jest „tym, co jest planowane”). Jeden działający program (proces) może składać się z wielu wątków. W różnych systemach można spotkać pojęcia pokrewne (np. *lightweight process* – LWP), ale nie zawsze są one synonimem wątku i bywają używane historycznie lub specyficznie dla danej rodziny systemów. (Patrz: Microsoft: About Processes and Threads; Oracle Solaris: Threads and LWPs)

W skład wątku wchodzą:

- licznik rozkazów;
- zbiór rejestrów;
- obszar stosu.

Wątek współdzieli z innymi równorzędnymi wątkami **zasoby procesu**, czyli m.in. sekcję kodu i danych oraz takie zasoby systemu operacyjnego jak otwarte pliki. Innymi słowy: wątki są „strumieniami wykonania” wewnątrz jednego procesu Microsoft: About Processes and Threads.

Aby zrozumieć pojęcie wątku, najlepiej porównać go z procesem tradycyjnym, tzw. *ciężkim* (ang. *heavyweight*). Najprostszy przypadek to proces posiadający tylko jeden wątek - wtedy proces i wątek „zachowują się” podobnie, choć nadal proces ma własną przestrzeń adresową i zasoby, a wątek jest tylko jednostką wykonania.

Wątki, w przeciwieństwie do procesów, można zwykle przełączać taniej, tj. z mniejszym użyciem czasu procesora. Przełączenie wątku nadal wymaga przełączenia zbioru rejestrów i stosu, jednak w ramach jednego procesu nie trzeba przełączać całej przestrzeni adresowej, co zazwyczaj upraszcza i przyspiesza operację.

Porównując wątki i procesy, rozważmy sposób sterowania wieloma wątkami i wieloma procesami. Każdy z procesów działa niezależnie, posiada własny licznik rozkazów, wskaźnik stosu oraz przestrzeń adresową. W przypadku, gdy zadania wykonywane przez procesy nie są ze sobą powiązane, wszystko jest w porządku. Jednakże, gdy mamy do czynienia z jednym zadaniem wykonywanym przez kilka procesów - marnowane są zasoby systemu (każdy proces przechowuje osobne kopie tych samych informacji). Proces wielowątkowy natomiast, wykonujący jedno zadanie, zużywa mniej zasobów (wątki korzystają z tego samego zbioru informacji).

Wątki są podobne do procesów pod wieloma względami. Tak jak procesy, wątki mogą znajdować się w jednym z określonych stanów: gotowości, zablokowania, aktywności i zakończenia. Aktywny może być tylko jeden wątek na *pojedynczym rdzeniu/logicznym CPU*, natomiast w komputerach wielordzeniowych równolegle aktywnych może być wiele wątków (po jednym na każdy rdzeń/hardware thread). Każdy wątek ma własny licznik rozkazów i stos, a jego wykonanie przebiega sekwencyjnie. Wątki mogą tworzyć wątki potomne i mogą blokować się do czasu zakończenia operacji wejścia-wyjścia - możliwa jest więc sytuacja działania jednego wątku, gdy zablokowany jest inny.

Różnicą pomiędzy wątkiem a procesem jest fakt, że wątki w ramach jednego procesu są od siebie zależne: każdy wątek ma dostęp do danych innego wątku (do każdego adresu w przestrzeni procesu). Niemożliwa jest więc sprzętowa izolacja pamięci między wątkami w obrębie jednego procesu. To z jednej strony zaleta (łatwe współdzielenie danych), ale z drugiej źródło problemów: błędy i brak synchronizacji w jednym wątku mogą uszkodzić dane widoczne dla innych wątków (do mechanizmów synchronizacji wrócę w dalszej części podręcznika).

Po co wątki?

Wątki stosuje się przede wszystkim wtedy, gdy jedno zadanie (jeden program) ma wykonywać kilka czynności „naraz” lub przynajmniej sprawiać takie wrażenie. Ponieważ wątki współdzielą pamięć procesu, łatwiej jest im wymieniać dane niż osobnym procesom, a przełączanie wątków bywa tańsze niż przełączanie procesów.

Typowe, intuicyjne zastosowania wątków to:

- zachowanie responsywności programu - np. wątek interfejsu użytkownika reaguje na mysz i klawiaturę, a inny wątek wykonuje dłuższe obliczenia lub operacje na plikach;
- obsługa wielu niezależnych zdarzeń - np. program sieciowy może równolegle obsługiwać komunikację i przetwarzanie danych;
- wykorzystanie wielu rdzeni procesora - podział jednego dużego zadania na części, które mogą wykonywać się równolegle;
- tzw. „pipeline” - jeden wątek pobiera dane, drugi je przetwarza, a trzeci zapisuje lub wysyła dalej.

Wątki w jądrze i wątki w przestrzeni użytkownika

Różne systemy operacyjne w różny sposób traktują wątki. Popularne są dwie koncepcje:

- wątki są obsługiwane przez jądro - system zawiera zbiór funkcji do obsługi wątków i to jądro planuje ich wykonanie;
- wątki są tworzone powyżej jądra systemu za pomocą funkcji bibliotecznych wykonywanych z poziomu użytkownika (ang. *user-level threads*).

Ponieważ wątki tworzone na poziomie użytkownika nie wymagają każdorazowego przejścia do jądra (korzysta się z nich za pomocą wywołań bibliotecznych), mogą być szybciej przełączane. Może jednak zdarzyć się sytuacja, w której po wykonaniu blokującego wywołania systemowego cały proces musi czekać, ponieważ jądro planuje tylko procesy i „nie wie”, że w środku procesu istnieje wiele wątków. Dlatego

też w systemach ogólnego przeznaczenia najczęściej stosuje się rozwiązania, w których wątki są widoczne dla jądra i mogą być planowane niezależnie. (Źródło: Microsoft: About Processes and Threads; Oracle Linux: POSIX Threads)

Modele implementacji wątków

Różnice opisane wyżej często opisuje się w postaci modeli mapowania wątków „bibliotecznych” na jednostki planowane przez jądro. Najczęściej spotkacie trzy podejścia:

- **model 1:1** - każdy wątek utworzony w programie ma odpowiednik po stronie jądra; dzięki temu jądro może planować wątki niezależnie i wykonywać je równolegle na wielu rdzeniach;
- **model N:1** - wiele wątków programu jest zarządzanych wyłącznie w przestrzeni użytkownika i jądro widzi je jako jeden proces; przełączanie może być bardzo szybkie, ale blokujące wywołanie systemowe może zatrzymać całe zadanie;
- **model M:N** - kompromis: wiele wątków użytkownika jest mapowanych na pewną liczbę jednostek planowanych przez jądro; daje to potencjalnie dobre własności, ale jest bardziej skomplikowane w implementacji.

W praktyce współczesne systemy ogólnego przeznaczenia najczęściej realizują wariant zbliżony do modelu 1:1 (czyli wątki są planowane przez jądro), bo umożliwia to dobre wykorzystanie wielordzeniowych procesorów i przewidywalne działanie w obliczu blokujących operacji wejścia-wyjścia. (Źródło: Oracle Linux: POSIX Threads; Microsoft: About Processes and Threads)

1.4. Synchronizacja

W tym rozdziale będę mówił głównie o **wątkach**, bo to one najczęściej współdzielą pamięć w ramach jednego procesu i dlatego najszybciej widać, po co potrzebna jest synchronizacja. Te same problemy i te same idee dotyczą również **procesów** zawsze wtedy, gdy współdzielą jakiś zasób (np. plik, urządzenie, gniazdo komunikacyjne, pamięć dzieloną).

Przetwarzanie danych w systemie współbieżnym polega na wykonywaniu instrukcji różnych wątków (lub procesów) - to przetwarzanie jest naprzemienne lub w pełni współbieżne (w systemach z więcej niż jednym rdzeniem) - lecz w obu przypadkach instrukcje powodują modyfikację danych, i w rezultacie - modyfikację systemu jako całości. Często jest to problem pomijalny - gdy przetwarzane dane należą tylko i wyłącznie do jednego procesu o jednym wątku. Lecz co w przypadku gdy wątki (albo procesy) chcą się nawzajem ze sobą komunikować?

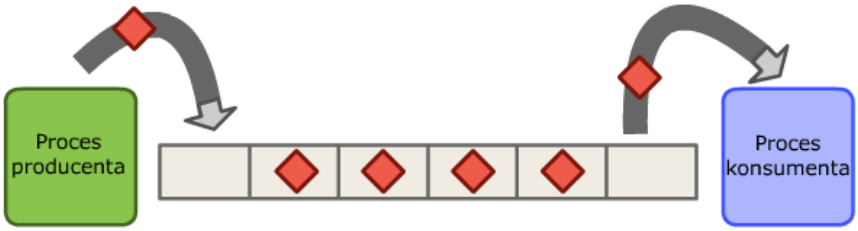
W przypadku konieczności współpracy musimy zawsze zapewniać, że współdzielone zasoby (najczęściej dane) pozostaną w spójnym stanie. Dane pozostają w spójnym stanie wtedy, gdy ich pojedynczą „paczkę" może w danej chwili modyfikować tylko jeden wątek, i żaden z innych wątków nie powinien jej w tym czasie modyfikować. Skoro tak - to potrzebne są jakieś mechanizmy zapewniające spójność danych. Postaram się omówić tutaj w skrócie kilka najbardziej popularnych. Ale zacznijmy od przykładowego algorytmu współpracy:

Algorytm producent / konsument

Zagadnienie producenta-konsumenta stanowi jeden z możliwych modeli współpracy. Zakładamy, że wątek producenta wytwarza dane, które konsumuje wątek konsumenta. Aby było to możliwe musi istnieć bufor (wyobraźmy sobie go jako tablicę) jednostek, który będzie zapełniany przez producenta i opróżniany przez konsumenta. W takim układzie, gdy producent produkuje jednostkę, konsument może wyciągnąć z bufora inną. Ważne jest jednak, aby producent i konsument podlegały synchronizacji tak, by konsument nie „wyciągał" jednostek jeszcze niewyprodukowanych, a w przypadku pustego bufora poczekał na wyprodukowanie czegoś do „wyciągania".

Komentarz

W praktyce producent i konsument mogą być wątkami w jednym procesie (wtedy bufor jest zwykłą strukturą w pamięci), ale mogą też być osobnymi procesami - o ile mają wspólny mechanizm komunikacji (np. pamięć dzieloną, kolejkę komunikatów, gniazda). Idea synchronizacji pozostaje ta sama.



Producent, bufor oraz konsument

Komentarz

Taki typ bufora, jak widzicie w tym przykładzie, w programowaniu nazywa się kolejką FIFO (*First In - First Out*, pierwszy wchodzi - pierwszy wychodzi).

Problem producenta-konsumenta może być dwojakiego rodzaju:

- z nieograniczonym buforem (ang. *unbounded-buffer*) - możliwa jest sytuacja oczekiwania przez konsumenta na nowe jednostki, jednakże producent produkuje je nieustannie;
- z ograniczonym buforem (ang. *bounded-buffer*) - możliwa jest sytuacja oczekiwania przez konsumenta na nowe jednostki, ale możliwe jest również, że czekał będzie producent w wyniku zapełnienia bufora.

Poniżej znajdziecie zapis algorytmu producenta-konsumenta w tzw. pseudokodzie, czyli uproszczonym języku programowania o składni zbliżonej do Pascal-a. Rzeczywiste implementacje są bardziej skomplikowane, lecz ten zapis powinien pozwolić Wam zrozumieć ideę działania takiego algorytmu. Przedstawiłem rozwiązanie problemu producenta-konsumenta z ograniczonym buforem, który używa pamięci dzielonej. W algorytmie tym producent i konsument korzystają z następujących wspólnych zmiennych:

```
var n;                              // rozmiar bufora
type jednostka=...;  // jednostka do konsumpcji
var bufor: array [0..n-1] of jednostka; // bufor jako tablica cykliczna
var we, wy: 0..n-1;  // wskaźniki do wejścia i wyjścia bufora
```

Zmienna *we* wskazuje na następne wolne miejsce w buforze. Gdy bufor jest pusty wskaźniki *we* i *wy* pokrywają się. Gdy bufor jest pełny to prawdziwy jest warunek *we+1* **mod** *n* = *wy* (*mod* oznacza resztę z dzielenia).

Poniżej przedstawione są kody producenta i konsumenta. Warto jeszcze wspomnieć iż instrukcja *nic* (ang. *NOP*) jest instrukcją pustą tzn. jej wykonanie nie daje żadnego efektu. Jednostki nowo produkowane znajdują się w zmiennej lokalnej *nowyprodukt*, zaś jednostki do skonsumowania są przechowywane w zmiennej *nowakonsumpcja*.

Producent:

```
repeat
    ...
  // produkuj jednostkę w nowyprodukt
  ...
    while we+1 mod n=wy do nic;
    bufor[we]:=nowyprodukt;
    we:=we+1 mod n;
until false
```

Konsument:

```
repeat
    while we=wy do nic;
  nowakonsumpcja:=bufor[wy];
  wy:=wy+1 mod n;
  ...
  // konsumuj jednostkę z nowakonsumpcja
  ...
until false
```

Podany algorytm zakłada, że wszelkie operacje wykonywane na pamięci dzielonej są tak zwanymi **operacjami atomowymi** - to oznacza, iż w momencie wykonywania takiej operacji wątek nie może zostać przerwany. Jeśli mógłby zostać przerwany - to oznaczałoby, że np. paczka wyprodukowana przez producenta jest tylko w połowie umieszczona w buforze (reszta to śmieci). Wtedy producent zostałby wywłaszczony, konsument wznowiony - i zacząłby pobierać połowiczne dane jako kompletną całość.

I tak doszliśmy do momentu, w którym czasem coś w programie komputerowym nie działa, ale zdarza się to tylko czasami, i nikt nie wie dlaczego tak się zdarza ... ;)

Synchronizacja

Wracając do głównego zagadnienia niniejszej części - skoro w systemie komputerowym możemy uruchomić więcej niż jeden wątek (lub proces), i każdy z nich ma prawo do korzystania ze wspólnych zasobów (przyjmijmy jako przykład zapis danych na dysk albo modyfikację wspólnej struktury w pamięci), to w jaki sposób zapewnić, że operacja wykonywana przez wątek A nie zostanie przerwana w „złym" momencie? I że wątek B nie zniszczy naszych częściowo zapisanych danych?

Unikanie takich problemów jest możliwe dzięki mechanizmom synchronizacji wątków (a w szczególnych przypadkach również procesów).

Definicja

Celem synchronizacji jest kontrola przepływu sterowania między wątkami tak, by dopuszczalne stały się tylko przeploty instrukcji dopuszczone przez programistę (by nie zachodziła interferencja w dostępie do współdzielonych danych i zasobów).

Mechanizm powstrzymywania wątków przed dostępem do danego zasobu jeżeli jest on używany przez jakikolwiek inny wątek, nazywa się **wzajemnym wykluczaniem** (mutual exclusion) a zasoby chronione – **zasobami krytycznymi**. Blokowanie to nie może odbywać się w pełni automatycznie (tzn. przez system operacyjny) - realizacja automatycznego mechanizmu wykluczeń jest zbyt trudna, choć istnieją języki programowania które takie automatyczne blokady umożliwiają (np. Java) kosztem niższej wydajności. System operacyjny udostępnia jednakże mechanizmy blokowania i kontroluje je. Odwołanie do zasobu krytycznego w wątku odbywa się w tzw. **sekcji krytycznej** (critical section). Mechanizm sekcji krytycznych musi być zaimplementowany w kodzie programu, a realizowany jest najczęściej poprzez użycie udostępnianych przez system operacyjny metod synchronizacyjnych.

Semafory

Przy omawianiu złożonych zagadnień synchronizacji wygodnie jest posługiwać się narzędziem synchronizacji nazywanym **semaforem** (ang. *semaphore*). Semafor jest strukturą zawierającą zmienną typu całkowitego, której można nadać wartość początkową i do której można się odwoływać tylko za pośrednictwem dwu niepodzielnych (atomowych) operacji: P (hol. *proberen*, ang. *wait*, pol. *czekaj*) oraz V (hol. *verhogen*, ang. *signal*, pol. *sygnalizuj*). Niepodzielność tych instrukcji jest zapewniana przez system operacyjny, często przy wsparciu procesora.

Definicja semafora przedstawia się następująco:

```
1.  poczekaj(S):
2.      while S<=0 do nic;
3.      S:=S-1;
4.    ... wykonuj zadania ...
5.
6.  zasygnalizuj(S):
7.     S:=S+1;
```

Ważne jest, aby zmiany wartości semafora wykonywane za pomocą operacji *poczekaj* i *zasygnalizuj* uniemożliwiały sytuację, w której dwa wątki próbowałyby modyfikować wartość semafora. Dodatkowo, podczas operacji *poczekaj(S)*, nie może wystąpić przerwanie w trakcie sprawdzania wartości zmiennej S (`dopóki S<=0`) oraz jej aktualizacji.

Mutexy

Semafory stosuje się często przy rozwiązywaniu problemów dostępu do sekcji krytycznej, której przydzielone jest kilka wątków. Wątki te korzystają ze wspólnego semafora nazywanego w tym przypadku **mutex** (ang. *mutual exclusion*), zapewniającego wzajemne wykluczanie tak, aby zapewnić dostęp tylko jednemu spośród nich. Organizacja każdego z wątków korzystających z semafora przedstawia się następująco:

```
1.  repeat
2.      poczekaj(mutex);
3.      ...
4.      sekcja krytyczna
5.      ...
6.      zasygnalizuj(mutex);
7.      ...
8.      reszta działań
9.      ...
10. until false;
```

Semafory stosujemy także przy rozwiązywaniu problemów związanych z synchronizacją. Jeżeli na przykład chcielibyśmy, aby dwa wątki T-X i T-Y, zawierające odpowiednio rozkazy S-X i S-Y, wykonały się sekwencyjnie, tzn. w kolejności X, potem Y. Możliwa jest prosta realizacja takiego problemu, gdy wprowadzimy wspólną dla obu wątków zmienną *synchronizacja*, nadawszy jej wartość początkową "0". Zakładając, że dołączymy do wątków T-X i T-Y instrukcje:

```
1.  T-X:
2.      S-X;
3.      sygnalizuj(synchronizacja);
```

```
1.  T-Y:
2.      czekaj(synchronizacja);
3.      S-Y;
```

możliwe stanie się wykonanie najpierw kodu S-X, później S-Y, ponieważ dopóki zmienna *synchronizacja* wynosi "0", semafor *czekaj* blokuje wykonanie instrukcji S-Y. Wykonanie S-Y możliwe staje się dopiero po wykonaniu rozkazu *sygnalizuj(synchronizacja)*.

Implementacja

Obsługa semaforów może wymagać **aktywnego czekania** (ang. *busy waiting*). Polega to na tym, że wątek chcąc dostać się do sekcji krytycznej, zajmowanej przez inny wątek, zmuszony jest do wykonywania pętli w sekcji wejściowej. Aktywne czekanie marnuje czas procesora, który mógłby wykonywać nieco ważniejsze zadania. Semafory tego typu nazywa się też **wirującą blokadą** (ang. *spinlock*).

Wirujące blokady w przypadku założenia, że czas oczekiwania pod blokadą jest bardzo krótki, stają się bardzo użyteczne w środowiskach wieloprocesorowych. Ich zaletą jest fakt, że postój pod semaforem nie wymaga przełączania kontekstu, który może trwać długo.

Nie trzeba jednak traktować aktywnego czekania jako niezniszczalnego. Aktywne czekanie można bowiem wyeliminować. Aby było to możliwe, wątek, który musiałby czekać aktywnie, powinien mieć możliwość zablokowania się (ang. *block*). Zablokowanie się wątku powoduje umieszczenie go w kolejce do danego semafora oraz zmianę stanu na "czekanie", po czym wybierany jest do wykonania kolejny wątek. Wznowienie działania zablokowanego wątku powoduje sygnał *zasygnalizuj* pochodzący od innego wątku. Wykonywana jest wtedy operacja budzenia (ang. *wakeup*), która zmienia stan na "gotowość". Powoduje to przejście do kolejki gotowych do wykonania przez procesor.

Semafor, o którym tak już dużo napisałem można zrealizować jako prosty rekord:

```
type semaphore = record
    wartosc: integer;
    L: list of proces;
end;
```

Jak widzimy semafor składa się z dwóch pól: wartości całkowitej i listy „czekających” (w praktyce mogą to być zarówno procesy, jak i wątki - zależnie od tego, co synchronizujemy). Operacje na tak zdefiniowanym semaforze można przedstawić następująco:

```
poczekaj(S):
    S.wartosc:=S.wartosc-1;
    if S.wartosc<0 then begin
        dodaj proces do S.L;
        blokuj;
    end;
```

```
zasygnalizuj(S):
    S.wartosc:=S.wartosc+1;
    if S.wartosc<=0 then begin
        usuń proces P z S.L;
        obudź(P);
    end;
```

Tak zaimplementowane operacje *poczekaj* i *zasygnalizuj* nie eliminują całkowicie efektu aktywnego czekania, jednakże zapewniają usunięcie tego zjawiska z wejść do sekcji krytycznych programów użytkowych. Ma to ogromne znaczenie, gdyż sekcje krytyczne tych programów mogą zajmować dużo czasu lub być często zajęte. Aktywne czekanie w takich warunkach mogłoby być fatalnym posunięciem i zakończyłoby się dużym spadkiem wydajności systemu.

Zdarzenia

Zdarzenia (events) są kolejnymi obiektami mogącymi służyć do synchronizacji. Ich ogólnym przeznaczeniem jest komunikacja między wątkami (albo procesami) - a dokładnie przekazywanie sygnałów typu „coś się stało” bez konieczności ciągłego odpytywania (busy waiting). Najczęstszy schemat wygląda tak: jeden wątek *czeka* na zdarzenie (blokuje się, aż warunek zostanie spełniony), a inny wątek w pewnym momencie *sygnalizuje* zdarzenie (informuje „możesz kontynuować”). W praktyce zdarzenia są wygodne do budowania prostych zależności: „producent przygotował dane”, „zadanie w tle skończyło się”, „można bezpiecznie wejść do kolejnego etapu”.

W systemie Microsoft Windows istnieje bezpośrednie pojęcie **obiektu zdarzenia** (event object). Jeden z elementów tworzy lub otwiera zdarzenie (np. CreateEvent lub OpenEvent), a inne wątki/procesy mogą na nie czekać funkcjami typu WaitForSingleObject / WaitForMultipleObjects. Zdarzenie może być **automatycznie kasowane** (auto-reset) - wtedy pojedyncza sygnalizacja „budzi” jeden czekający wątek i od razu wraca do stanu niesygnalizowanego - albo **ręcznie kasowane** (manual-reset) - wtedy po sygnalizacji pozostaje „włączone”, dopóki ktoś go nie skasuje (ResetEvent), co pozwala obudzić wielu czekających. Dzięki temu zdarzenia świetnie pasują do scenariuszy typu „wystartuj dopiero gdy inicjalizacja zakończona” albo „obudź wątek roboczy, gdy pojawi się praca do wykonania”.

W systemach z rodziny Unix (POSIX) zamiast jednego, uniwersalnego „event object” spotyka się kilka popularnych mechanizmów realizujących ten sam cel. Dla wątków w jednym procesie typowym narzędziem jest **zmienna warunkowa** (pthread_cond_t) używana razem z **muteksem**: wątek czekający wykonuje pthread_cond_wait, a wątek sygnalizujący wywołuje pthread_cond_signal (budzi jeden) lub pthread_cond_broadcast (budzi wszystkich). Dla procesów częstym odpowiednikiem są np. **semafory POSIX** (sem_wait / sem_post), **potoki** (pipe) lub **gniazda** - gdzie „zdarzeniem” może być pojawienie się danych do odczytu (czekamy wtedy np. przez select/poll/epoll). Innymi słowy: w Unix-ach idea zdarzeń zwykle realizowana jest przez prymitywy „czekaj aż warunek będzie spełniony” albo „czekaj aż pojawią się dane”, zamiast przez jeden, nazwany obiekt zdarzenia jak w Windows.

Warto podkreślić, że zdarzenia (w sensie mechanizmu budzenia) bywają używane także do ochrony danych, ale najczęściej w połączeniu z muteksem: muteks chroni spójność modyfikacji, a zdarzenie/zmienna warunkowa służy do efektywnego czekania na moment, gdy dane osiągną pożądany stan (np. „bufor nie jest pusty” albo „bufor nie jest pełny”).

Zakleszczenie i głodzenie

Nieodpowiednie wykorzystywanie semaforów, mutex-ów i im podobnych elementów może prowadzić do nielubianych przez programistów sytuacji, które pospolici użytkownicy komputerów nazywają "powieszeniem się" aplikacji. Mowa tutaj o zakleszczeniu (ang. *deadlock*), czyli stanie, gdy każdy wątek (lub proces) w jakimś zbiorze oczekuje na zdarzenie, które może być spowodowane tylko przez inny element z tego zbioru. Dla semaforów zdarzeniem tym jest operacja *zasygnalizuj*.

Przykładem takiego zdarzenia są procesy Px i Py, które korzystają z semaforów A i B, o wartościach "1", których postać przedstawia się następująco:

```
1.  Px:
2.
3.  poczekaj(A);
4.  poczekaj(B);
5.  .
6.  .
7.  .
8.  zasygnalizuj(A);
9.  zasygnalizuj(B);
```

```
1.  Py:
2.
3.  poczekaj(B);
4.  poczekaj(A);
5.  .
6.  .
7.  .
8.  zasygnalizuj(B);
9.  zasygnalizuj(A);
```

W takiej sytuacji powstaje zakleszczenie, gdyż proces Px musi czekać na wykonanie operacji *zasygnalizuj(A)* przez proces Py, zaś Proces Py czeka na wykonanie przez Px rozkazu *zasygnalizuj(B)*. I tak będą czekać w nieskończoność ... podobnie jak użytkownik tak napisanej aplikacji.

Kolejnym problemem związanym z semaforami jest głodzenie, czyli konieczność nieskończonego oczekiwania pod semaforem. Problem ten jest wynikiem niskiego priorytetu danego procesu (lub wątku). Element jest ignorowany np. w wyniku dużego natłoku innych zadań, które mają wyższe priorytety. Najczęściej głodzenie pojawia się w przypadku użycia kolejki LIFO (ang. *Last In First Out*) jako listy oczekujących do semafora.

Przydział zasobów i zakleszczanie

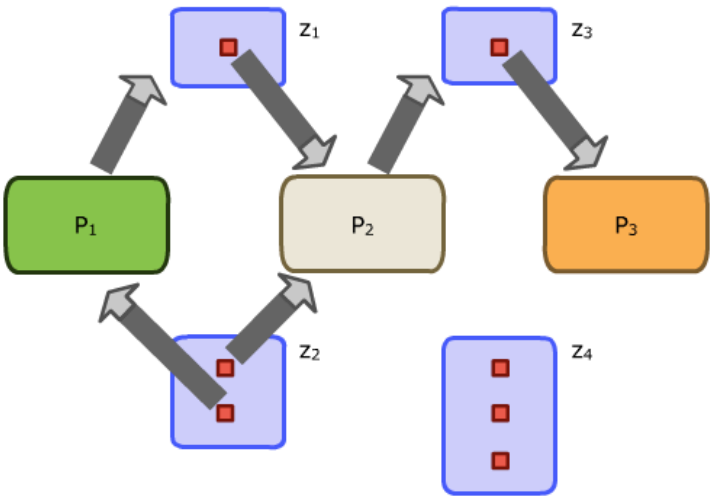
W przypadku próby uzyskania dostępu do współdzielonych zasobów dość często występuje problem zakleszczania. Ba - nie będzie przesadą stwierdzenie, że - obok zarządzania pamięcią - zakleszczanie jest największą zmorą programistów. Poniżej znajdziecie opis kilku metod, które pozwalają radzić sobie z tym nieprzyjemnym zjawiskiem.

Graf przydziału zasobów

Graf przydziału zasobów (ang. *system resource-allocation graph*) bardzo dobrze nadaje się do opisywania zakleszczeń. Składnikami tego skierowanego grafu są:

1. wierzchołki W, składające się z węzłów:
   - $P=\{P_1, P_2,..., P_n\}$ - wszystkie procesy systemu;
   - $Z=\{Z_1, Z_2,..., Z_n\}$ - wszystkie typy zasobów systemowych;
2. krawędzie K, które dzielimy na:
   - krawędzie skierowane od procesu P_x do zasobu Z_y (zapisujemy $P_x \rightarrow Z_y$) - oznacza to zamówienie zasobu Z_y przez proces P_x; krawędzie zwane są *krawędziami zamówienia* (ang. *request edge*);
   - krawędzie skierowane od zasobu Z_y do procesu P_x (zapisujemy $Z_y \rightarrow P_x$) - oznacza to przydzielenie zasobu Z_y do procesu P_x; krawędzie te nazywa się *krawędziami przydziału* (ang. *assignment edge*).

Ponadto zbiór procesów P na grafie przedstawia się w postaci kółek, zaś zbiór typów zasobów Z w postaci prostokątów. Każdy z typów zasobów Z_y może mieć więcej niż jeden egzemplarz, zaznaczany jako kropka w prostokącie zasobu. Podczas gdy krawędź zamówienia dochodzi tylko do boku prostokąta zasobu, krawędź przydziału rozpoczyna się od konkretnego egzemplarza (czytaj kropki) typu zasobu wewnątrz prostokąta.



Graf przydziału zasobów

Sposób działań systemu odpowiadających zmianom krawędzi jest następujący:

1. Zamówienie przez proces P_x zasobu Z_y powoduje pojawienie się krawędzi zamówienia.
2. Realizacja zamówienia powoduje zamianę krawędzi zamówienia w krawędź przydziału.
3. Brak jakiejkolwiek krawędzi oznacza, że proces zwolnił już niepotrzebny mu zasób.

Czytając o grafie przydziału zasobów, na pewno zastanawiacie się: co to ma wspólnego z zakleszczeniami? Jeśli graf zawiera cykle (pętle) może dojść do zakleszczeń, jeżeli nie zawiera cykli sytuacja taka jest niemożliwa. A czym jest cykl? Na rysunku wyżej nie występuje żaden cykl. Jeżeli jednak wyobrazilibyśmy sobie sytuację, w której proces P3 zamawia zasób Z2 (pojawia się wtedy krawędź zamówienia P3 - Z2), to w tak zmodyfikowanym układzie występują dwa cykle:

- $P_1 \rightarrow Z_1 \rightarrow P_2 \rightarrow Z_3 \rightarrow P_3 \rightarrow Z_2 \rightarrow P_1$
- $P_2 \rightarrow Z_3 \rightarrow P_3 \rightarrow Z_2 \rightarrow P_2$.

Zakleszczeniem możemy określić sytuację, w której każdy z zasobów w cyklu ma tylko jeden egzemplarz. W przypadku natomiast, gdy któryś z zasobów w cyklu ma większą liczbę egzemplarzy niż 1 nie przesądza to o występowaniu zakleszczenia.

W przedstawionym powyżej grafie systemu, do zakleszczenia dojdzie na pewno, gdy proces P_3 zechce skorzystać z zasobu Z_2.

Przetrzymanie i oczekiwanie

Warunek przetrzymywania i oczekiwania jest spełniony, gdy istnieje proces, któremu został przydzielony co najmniej jeden zasób. Ponadto proces ten oczekuje na przydział kolejnego zasobu, który jest używany przez inny proces.

Zapobiegniemy zakleszczeniu wtedy i tylko wtedy, gdy uda nam się dokonać negacji powyższego warunku. Musimy więc doprowadzić do sytuacji, w której proces zamawiający zasób nie posiada sam żadnych zasobów. Opiszę pokrótce dwa protokoły realizujące takie założenie:

1. Należy umieścić wywołania funkcji systemowych, opisujących zamówienia zasobów dla procesu, za wywołaniami wszystkich innych funkcji systemu.
2. Umożliwić procesowi zamawianie zasobów wtedy i tylko wtedy, gdy proces nie posiada żadnych zasobów (przed zamówieniem nowych zasobów proces musi oddać wszystkie zasoby, z których w danej chwili korzysta).

Podstawową wadą obu powyższych rozwiązań jest to, że *wykorzystanie zasobów* może być zaskakująco małe, gdyż niemożliwe będzie korzystanie z dużej części przydzielonych zasobów przez długi czas. Drugą z wad opisanych protokołów jest możliwość wystąpienia głodzenia. Proces, który chce skorzystać z "rozchwytywanego" zasobu, może czekać do nieskończenie długo tylko dlatego, że zasób ten będzie ciągle w posiadaniu innego procesu.

Czekanie cykliczne

Warunek czekania cyklicznego występuje wtedy i tylko wtedy, gdy istnieje zbiór procesów czekających $P=\{P_0, P_1, ..., P_n\}$, takich że P_0 oczekuje na zasób będący w posiadaniu procesu P_1, P_1 zaś czeka na zasób, który jest w posiadaniu procesu P_2, ..., a P_n oczekuje na zwolnienie zasobu przez proces P_0.

Najbardziej popularnym sposobem uniknięcia czekania cyklicznego jest uporządkowanie wszystkich typów zasobów i dopilnowanie, by każdy z procesów otrzymywał te zasoby rosnąco. Przyporządkowujemy więc jednoznacznie każdemu zasobowi należącemu do zbioru $Z=\{Z_0, Z_1, ..., Z_i\}$ liczbę całkowitą (w celu określenia rosnącego porządku procesów). Na początku proces może zamówić dowolną liczbę egzemplarzy wybranego typu zasobu. Potem proces może zamawiać jedynie egzemplarze tych typów, które mają większe numery. Jeśli na przykład na początku proces zamówił zasób Z_2, to później musi zamawiać zasoby $Z_x > Z_2$. Należy zaznaczyć przy tym, że możliwe jest zamówienie kilku egzemplarzy tego samego typu z użyciem jednego tylko zamówienia.

Typowe problemy synchronizacji

Algorytm producent/konsument w wersji ogólnej

Wróćmy do przykładu producenta - konsumenta z początku tej lekcji. Jeżeli przyjmiemy, że mamy *n* buforów, a każdy z nich mieści 1 jednostkę, oraz że semafor *mutex* (umożliwiający wzajemne wykluczanie dostępu do buforów) jest na starcie równy 1, zaś semafory *pusty* i *pełny* zawierają informację o liczbie pustych (wartość początkowa n) i pełnych (wartość początkowa 0) buforów, to ogólna wersja algorytmu wygląda tak:

```
1.  repeat
2.      ...
3.      produkuj jednostkę w nowyprodukt
4.      ...
5.      poczekaj(pusty);
6.      poczekaj(mutex);
7.      ...
8.      dodaj jednostkę nowyprodukt do bufora bufor
9.      ...
10.     zasygnalizuj(mutex);
11.     zasygnalizuj(pełny);
12. until false
```

Producent ...

```
repeat
    poczekaj(pełny);
    poczekaj(mutex);
    ...
    wyjmij jednostkę z bufora bufor do nowakonsumpcja
    ...
    zasygnalizuj(mutex);
    zasygnalizuj(pusty);
    ...
    konsumuj jednostkę z nowakonsumpcja
    ...
until false
```

... i konsument.

Instrukcje *poczekaj* oznaczają oczekiwanie z powodu faktu, że bufor jest pusty lub pełny. Instrukcja *zasygnalizuj* opisuje sytuację, w której bufor staje się pusty lub pełny, a także uruchomienie semafora *mutex*.

Problem czytelników i pisarzy

Problem ten odnosi się do środowisk, w których zasoby mogą być dzielone. W takich środowiskach może zachodzić sytuacja dostępu do danego zasobu przez kilka wątków jednocześnie. Niektóre z tych wątków odczytują tylko zawartość obiektu dzielonego, inne zapisują do niego informacje (aktualizacja). Wątki odczytujące dane z obiektu dzielonego to *czytelnicy*, zaś zapisujące dane to *pisarze*.

Problem czytelników i pisarzy (ang. *readers-writers problem*) polega na zapewnieniu "prywatności" pisarzowi. Nie może zdarzyć się sytuacja, w której dany obiekt współdzielony jest przez pisarza i inny wątek, czy to czytelnika, czy to innego pisarza - nie może istnieć sytuacja w której książka ma kilku autorów, oraz w której ktoś czyta nieskończone dzieło.

Problem ten ma kilka odmian. Najłatwiejsza z nich nazywana jest pierwszym problemem czytelników i pisarzy. Metoda ta zakłada, że żaden czytelnik nie musi czekać (pomimo tego, że pisarz jest gotowy) na zakończenie pracy pozostałych czytelników. W odniesieniu do świata informatyki - istnieje różnica w dostępie do zapisu i do odczytu. Dostęp do zapisu jest dostępem na wyłączność - nikt w tym czasie nie może pisać ani czytać. Dostęp do odczytu jest dostępem współdzielonym - ciągle nikt inny nie może pisać, ale czytać może wielu.

Kolejna odmiana problemu czytelników i pisarzy zakłada, że gotowość pisarza oznacza, że żaden nowy czytelnik nie będzie mógł współpracować z obiektem. Ponieważ oba przedstawione powyżej schematy mogą powodować głodzenie, stosuje się również inne wersje rozwiązania tego problemu.

Poniżej zostanie opisane rozwiązanie pierwszego problemu czytelników i pisarzy. Załóżmy, że wszystkie z wątków mają wspólne następujące zmienne:

```
var mutex, pisarz: semaphore; // wartość początkowa "1" dla obu semaforów
liczba_czytelnikow: integer;  // wartość początkowa "0"
```

Przed napisaniem struktury czytelnika warto jeszcze nadmienić, że semafor *mutex* zajmuje się wzajemnym wykluczaniem podczas nadpisywania zmiennej *liczba_czytelnikow*. Semafor *pisarz* daje gwarancję wzajemnego wykluczania pisarzy, a ponadto jest używany przez pierwszy wątek wchodzący lub ostatni wątek wychodzący z sekcji krytycznej czytelnika. Semafor ten nie jest natomiast używany przez czytelników wchodzących/wychodzących podczas, gdy inni czytelnicy znajdują się w sekcjach krytycznych. Rozwiązanie naszego problemu.

Tak wygląda struktura pisarza:

```
poczekaj(pisarz)
...
teraz proces sobie pisze i pisze, i pisze ...
...
zasygnalizuj(pisarz);
```

Struktura czytelnika wygląda następująco:

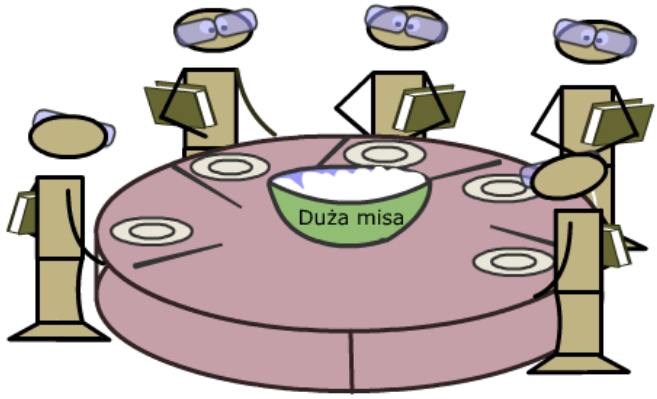
```
poczekaj(mutex);
liczba_czytelnikow:=liczba_czytelnikow+1;
if liczba_czytelnikow=1 then poczekaj(pisarz);
zasygnalizuj(mutex);
...
proces czyta sobie i czyta, i czyta ...
...
poczekaj(mutex);
liczba_czytelnikow:=liczba_czytelnikow-1;
if liczba_czytelnikow=0 then zasygnalizuj(pisarz);
zasygnalizuj(mutex);
```

Po wykonaniu przez pisarza operacji *zasygnalizuj(pisarz)* możliwe są dwie drogi: można wznowić działanie pojedynczego pisarza lub czekających czytelników, zależnie od wyboru planisty (scheduler-a).

Problem głodujących filozofów

Uwaga, wybieramy się na wycieczkę. Znajdujemy się teraz w miejscu bliżej nieokreślonym, w którym jest tylko jeden mebel - stół. Przy tymże stole siedzi pięć osób. Wszyscy mają przed sobą miseczki, a na środku stołu stoi misa z ryżem. Ponadto, na stole znajduje się pięć pałeczek rozmieszczonych pomiędzy miseczkami nieznanych nam osób. Po dokładniejszych oględzinach okazuje się, że ci ludzie są filozofami. A wiecie dlaczego? Ponieważ przez cały czas myślą, a gdy są głodni sięgają po pałeczki, aby zjeść trochę ryżu, potem znowu myślą. I tu dochodzimy do naszego problemu. Jak nasz drogi filozof ma wziąć dwie pałeczki, skoro sąsiad też może chcieć coś zjeść? Przecież filozof nie będzie wyrywał siłą koledze pałeczki z ręki.

Można by przypuszczać, że problem filozofów nie ma wiele wspólnego z synchronizacją - lecz tak nie jest. Problem obiadujących filozofów jest klasycznym problemem konieczności przydziału wielu zasobów do wielu procesów w warunkach zagrożenia głodzeniem i zakleszczeniem.



Pięciu współczesnych obiadujących filozofów

Przedstawię proste rozwiązanie tego problemu, oparte na semaforach, nie pozbawione jednak niebezpieczeństw. Operacja *poczekaj* oznacza podniesienie przez filozofa pałeczki, zaś operacja *zasygnalizuj* oznacza odłożenie pałeczki. Nasza pałeczka wygląda natomiast następująco:

```
1.  var paleczka: array[0..4] of semaphore; // wartość początkowa=1
```

Po wielu zmaganiach umysłowych dochodzimy do przykładowego algorytmu n-tego filozofa.

```
1.  repeat
2.      poczekaj(pałeczka[n]);
3.      poczekaj(pałeczka[n+1 mod 5]);
4.      ...
5.      jedzenie, jedzenie, jedzenie, np. ryżu
6.      ...
7.      zasygnalizuj(pałeczka[n]);
8.      zasygnalizuj(pałeczka[n+1 mod 5]);
9.      ...
10.     myślenie nie boli, więc filozof myśli
11.     ...
12. until false
```

Struktura procesu n-tego filozofa

Rozwiązanie przedstawione powyżej nie jest najlepsze. Daje co prawda gwarancję, że dwaj sąsiedzi nie będą jedli jednocześnie, jednakże w przypadku gdy każdy z filozofów w tym samym czasie weźmie pałeczkę (by było to możliwe każdy z filozofów musi wziąć pałeczkę leżącą po tej samej stronie, np. prawej) tak, że wszystkie pałeczki zostaną podniesione (elementy tablicy pałeczka są równe "0"), nie będzie możliwe podniesienie przez któregokolwiek z filozofów pałeczki drugą ręką. Zatrzymamy się w pętli nieskończonej - zakleszczymy się.

Naturalnie możliwa jest eliminacja takich zakleszczeń. Przykładowe sposoby eliminacji tego zagrożenia:

- do stołu zasiada czterech filozofów, podczas gdy jest wolnych pięć miejsc;
- filozof może podnosić pałeczki wtedy i tylko wtedy, gdy po prawej i po lewej stronie są one dostępne;
- wprowadzenie asymetrii - filozofowie nieparzyści zaczynają podnosić pałeczki z lewej strony, zaś parzyści ze strony prawej.

Spis treści

1. System plików
1.1. Zasada działania
1.2. Dostępne systemy plików

1. System plików

W trakcie tej lekcji chciałbym przedstawić Wam jeden z najważniejszych modułów systemu operacyjnego z punktu widzenia użytkownika – podsystem obsługi plików.

To od systemu plików w dużej mierze zależy szybkość działania pamięci masowych oraz bezpieczeństwo danych na nich umieszczonych. Dodatkowo, o ile użytkownik systemu operacyjnego raczej nie ma wpływu na algorytmy zarządzania pamięcią, kolejkowania procesów czy też zapobiegania zakleszczeniom (opisane w poprzedniej lekcji), o tyle najczęściej może sam zadecydować, jaki system plików ma wykorzystywać dla swojej stacji roboczej, czy też na serwerze, którym się opiekuje.

Mam nadzieję, że wiedza o właściwościach i możliwościach poszczególnych systemów plików pomoże w prawidłowym i świadomym wyborze.

Pliki i ich organizacja

Pojęcie pliku

Dane bezpośrednio potrzebne procesorowi do wykonywania jego zadań są umieszczane w pamięci operacyjnej systemu. Jest to jednak pamięć ulotna i dane w niej zapisane są tracone po wyłączeniu zasilania. Nie nadaje się więc ona do trwałego magazynowania informacji. Ponadto rozmiar pamięci operacyjnej jest ograniczony, dlatego większe zbiory danych przechowuje się na nośnikach trwałych.

W praktyce dane magazynuje się na różnych nośnikach takich jak dyski HDD, dyski SSD/NVMe (pamięć flash), pamięci USB i karty SD, a w zastosowaniach archiwizacyjnych także taśmy. Dyski optyczne spotyka się dziś rzadziej – głównie w zastosowaniach niszowych.

> **Definicja**
>
> Plik jest pojęciem logicznym i oznacza podstawową logiczną jednostkę magazynowania informacji. Jest to ciąg bajtów danych opatrzony nazwą i atrybutami. Z punktu widzenia użytkownika systemu operacyjnego plik jest całością, natomiast na fizycznym nośniku bywa przechowywany w innej formie (np. jako zestaw powiązanych ze sobą bloków). Sposób przechowywania plików, a także możliwe struktury, na jakie dzielony jest fizyczny nośnik, zależą od systemu plików (ang. *file system*).

Najważniejszą cechą pliku jest trwałość danych na nośniku, ale z punktu widzenia programisty kluczowy jest model dostępu do danych. Plik udostępnia strumień bajtów oraz bieżącą pozycję odczytu/zapisu (offset, czasem nazywany „kursorem"), mówiącą o tym, od którego miejsca będą czytane lub dopisywane dane. W wielu przypadkach pozycję tę można przestawiać i czytać/zapisywać dane od dowolnego miejsca w pliku (dostęp swobodny), choć sekwencyjne czytanie „od początku do końca" jest najczęstszym sposobem użycia.

> **Komentarz**
>
> W systemach z rodziny Unix/Linux idea „wszystko jest plikiem" nadal jest bardzo silna: także urządzenia, potoki czy gniazda są często udostępniane przez jednolity interfejs operacji wejścia/wyjścia. Różne obiekty mogą jednak mieć różne własności – przykładowo zwykły plik na dysku daje się „przewijać" (zmieniać offset), a potok (pipe) zazwyczaj nie. Zob. np. opis operacji zmiany offsetu: lseek(2).

Rodzaje i typy plików

Dane zapisane w pliku zależą od jego twórcy i mogą być bardzo różne. Ze względu na charakter zapisanych informacji pliki dzieli się na:

- programy;
- dane.

Dane zapisane w plikach mogą być różnego rodzaju. Mogą to być dane liczbowe, teksty, teksty z dodatkowymi informacjami o formatowaniu, obrazy grafiki komputerowej, nagrania dźwiękowe, zapisy sekwencji wideo itd.

Struktura pliku, czyli sposób jego wewnętrznej organizacji, zależy od rodzaju zapisanych informacji. Mówimy, że struktura pliku zależy od jego typu.

Przykładowe typy plików (często spotykane w środowiskach Windows) zostały zebrane w tabeli poniżej. Trzeba pamiętać, że to tylko konwencje – systemy operacyjne i programy mogą rozpoznawać typ pliku również na podstawie jego zawartości.

Typ pliku	Rozszerzenie nazwy	Funkcja
Wykonywalny	exe, com	Gotowy do uruchomienia program (np. w formacie PE)
Kod źródłowy	c, cpp, h, hpp, py, js	Kod źródłowy wyrażony w różnych językach
Skrypt / wsadowy	bat, cmd	Polecenia dla interpretera poleceń lub skrypt wykonywany przez interpreter
Tekstowy	txt, md	Dane i dokumenty tekstowe

Obraz	png, gif, jpg, bmp, tiff	Plik binarny zawierający obraz rastrowy
Archiwalny	zip, rar, 7z, tar	Plik spakowany odpowiednim algorytmem kompresji (czasem także archiwizacji)

Wewnętrzna organizacja struktury pliku jest uzależniona od typu pliku i od wymagań programu z niego korzystającego.

Z plikiem związanych jest kilka *atrybutów* (cech). Należą do nich m.in.:

- **nazwa** – ciąg znaków identyfikujących plik w sposób zrozumiały dla człowieka;
- **typ** – określa „rodzaj" przechowywanych informacji, na przykład: wykonywalny, wynikowy, kod źródłowy, tekstowy, archiwalny itd. W praktyce typ bywa rozpoznawany przez rozszerzenie, ale we współczesnych systemach nie jest to jedyne kryterium. W Unix/Linux o uruchomieniu programu decydują m.in. uprawnienia (bit wykonywania) oraz format pliku; w Windows istotny jest m.in. format programu (np. PE) i reguły uruchamiania. Zob. inode(7) oraz opis formatu PE: PE Format;
- **położenie** – pozwala na fizyczną lokalizację na nośniku;
- **rozmiar** – zawiera wielkość pliku najczęściej wyrażoną w bajtach;
- **ochrona** – informacje o tym kto i w jaki sposób może odczytywać lub modyfikować dane pliku;
- **daty** – np. data utworzenia, modyfikacji, ostatniego użycia.

W różnych systemach mogą występować różne zestawy atrybutów i różne ich znaczenie. Przykładowo w systemach Unix i pochodnych to, czy plik jest wykonywalny, wynika przede wszystkim z atrybutów uprawnień, a nie z rozszerzenia.

Operacje na plikach

Zazwyczaj to system operacyjny zapewnia mechanizmy dostępu i zarządzania plikami, choć nie jest powiedziane, że podobne mechanizmy nie mogą być również wbudowane w oprogramowanie użytkowe (np. sieciowe czy wirtualne systemy plików).

Każdy sensowny system operacyjny powinien zapewnić następujące operacje na pliku:

- **otwieranie i zamykanie** – uzyskanie identyfikatora/uchwytu pliku potrzebnego do dalszych operacji;
- **tworzenie** – utworzenie nowego, najczęściej pustego pliku, nadanie mu nazwy oraz udostępnienie do dalszych operacji;
- **odczyt** – pobranie danych z pliku do pamięci (zwykle od bieżącej pozycji odczytu);
- **zapis** – dopisanie lub nadpisanie danych w pliku (zwykle od bieżącej pozycji zapisu);
- **zmiana nazwy** – zmiana nazwy pliku bez konieczności kopiowania jego zawartości;
- **usuwanie** – usunięcie pliku z systemu plików;
- **skracanie** – usunięcie danych z pliku bez usuwania samego pliku (efektem może być pusty plik o długości 0);
- **przenoszenie** – zmiana położenia pliku w strukturze katalogów;
- **kopiowanie** – wykonanie kopii zawartości pliku pod inną nazwą lub w innym położeniu w strukturze katalogów;
- **odczyt i modyfikacja metadanych** – np. odczyt uprawnień, czasu modyfikacji, właściciela itp.

W różnych systemach operacyjnych mogą występować dodatkowe operacje na plikach.

Komentarz

Należy podkreślić, że plik jest strukturą logiczną, tak samo jak jego organizacja wewnętrzna. Zapis pliku na nośniku musi natomiast być zgodny z fizyczną organizacją pamięci masowej. Z tego powodu pliki są zwykle przechowywane w blokach (klastrach) o ustalonym rozmiarze.

Komentarz

W praktyce spotyka się dwa powiązane zjawiska: **straty blokowe** (czasem nazywane fragmentacją wewnętrzną), gdy ostatni blok pliku bywa tylko częściowo wykorzystany, oraz **fragmentację rozmieszczenia**, gdy kolejne części pliku leżą w wielu miejscach nośnika. Na dyskach HDD fragmentacja rozmieszczenia potrafi spowalniać odczyt; na SSD ma zwykle mniejsze znaczenie, choć nadal istnieją koszty organizacyjne.

1.1. Zasada działania

Pierwsze komputery pracujące w trybie wsadowym korzystały m.in. z kart perforowanych i taśm magnetycznych. Wraz z rozwojem systemów operacyjnych pojawiła się potrzeba trwałego przechowywania danych użytkowników – tak, aby były dostępne po zakończeniu programu, ponownym uruchomieniu komputera czy awarii zasilania. Współcześnie niemal każde urządzenie korzysta z jakiejś formy pamięci trwałej (SSD/NVMe, HDD, pamięć flash, karty SD/USB, a w archiwizacji także taśmy).

Narzędziem umożliwiającym organizację danych na nośnikach jest **system plików**. W praktyce nie jest to tylko „sposób ułożenia danych na dysku”, ale cały zestaw elementów: **format danych na nośniku** (metadane, katalogi, mapy zajętości, itp.), **kod w systemie operacyjnym** obsługujący ten format (sterownik systemu plików) oraz **interfejs** udostępniany programom (np. operacje otwórz/czytaj/zapisz). W wielu systemach (np. Unix/Linux) rolę ujednolicania obsługi różnych systemów plików pełni warstwa VFS (Virtual File System).

Głównym celem systemu plików (SP) jest przechowywanie danych w sposób zorganizowany, umożliwiający ich zapis, odnalezienie i odczyt. System plików zapewnia też operacje zarządzania plikami (kopiowanie, przenoszenie, usuwanie) oraz organizuje je w logiczną strukturę katalogów.

W typowej konfiguracji aplikacje nie operują bezpośrednio na surowych sektorach nośnika, tylko korzystają z wywołań systemowych (operacji na plikach). Bezpośredni dostęp do „surowego urządzenia” bywa możliwy wyłącznie dla kodu uprzywilejowanego (administrator, usługi systemowe) i jest ograniczany mechanizmami ochrony.

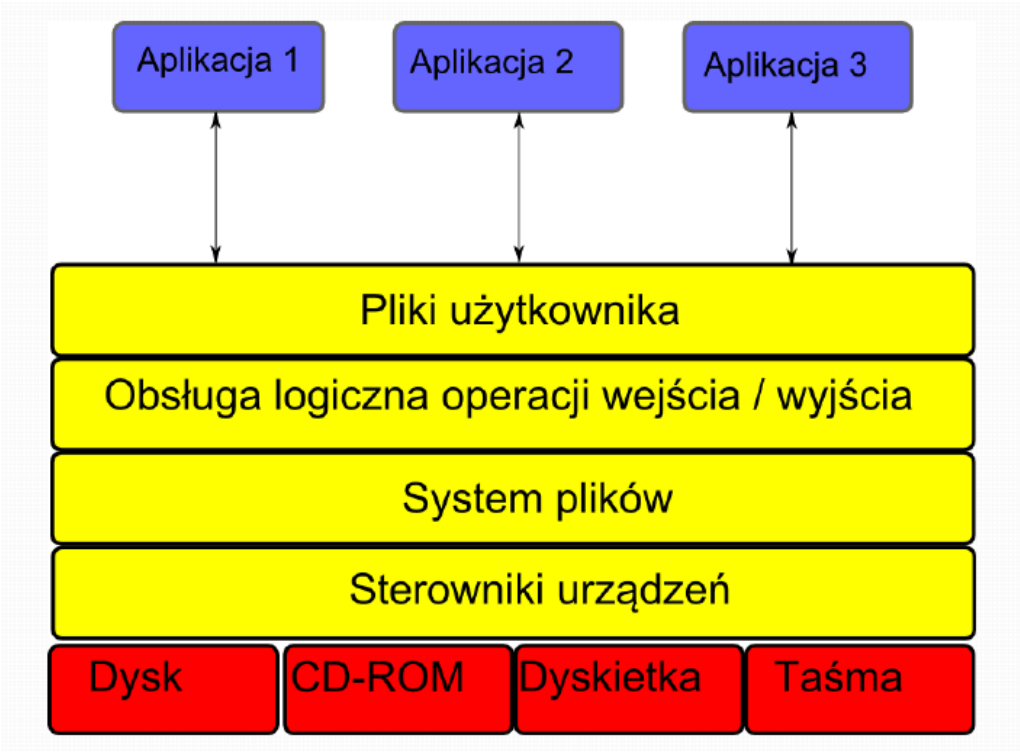
System plików jest jednym z najbardziej widocznych elementów systemu operacyjnego, bo to przez niego użytkownik widzi pliki i katalogi. W wielu systemach występuje pojęcie **montowania** (ang. *mount*) – czyli „podpinania” wolumenu (partycji) pod wybrany katalog w drzewie katalogów.

Głównym zadaniem systemu plików w systemie komputerowym jest zapewnienie prawidłowego i wydajnego dostępu do danych zapisanych na nośniku oraz zapewnienie ochrony przechowywanej informacji.

Ważnymi cechami SP są:

- długoterminowość (przechowywanie danych po wylogowaniu się użytkownika, wyłączeniu zasilania itp.);
- możliwość wykorzystywania danych przez różne procesy (a często także różne komputery – w przypadku systemów sieciowych);
- możliwość organizacji plików w struktury odzwierciedlające wzajemne relacje (np. drzewo katalogów).

System plików wraz z biblioteką funkcji służących do jego obsługi można przedstawić za pomocą schematu warstwowego jak poniższym rysunku:



Schemat warstwowy systemu plików

Programy użytkowe komunikują się z systemem plików przez wywołania systemowe (np. otwórz/czytaj/zapisz). Warstwa systemu plików tłumaczy nazwę i ścieżkę na metadane (np. wpis katalogu, identyfikator pliku), a następnie na konkretne bloki danych na nośniku. Dodatkowo system operacyjny zwykle buforuje dane w pamięci RAM (cache), aby ograniczać liczbę wolnych operacji dyskowych. Najniżej działa sterownik urządzenia, który komunikuje się z kontrolerem dysku/SSD.

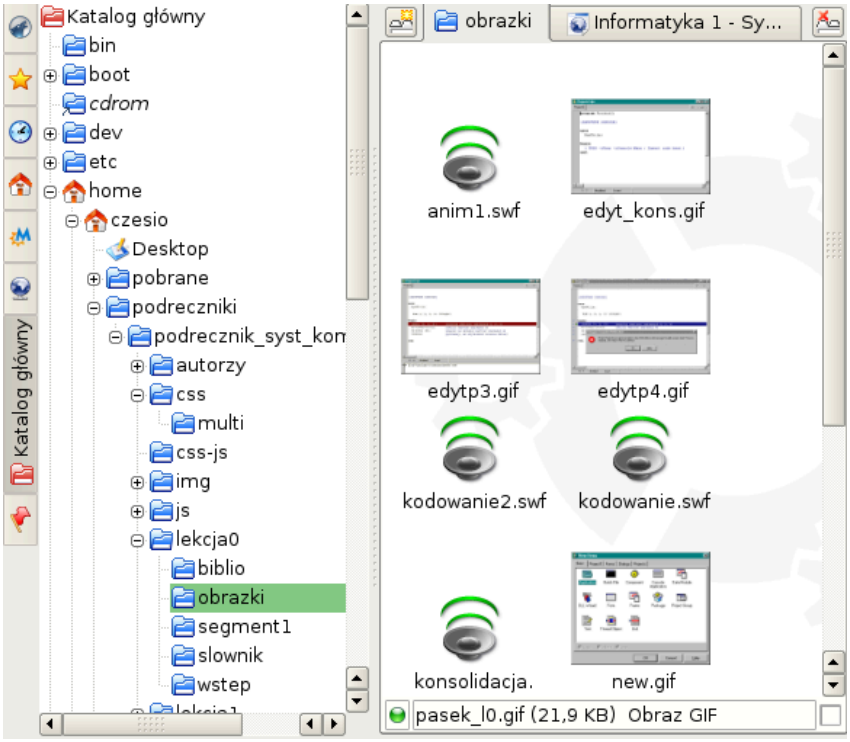
Zazwyczaj pliki organizowane są w grupy nazywane katalogami.

Struktura katalogów

Katalog określa położenie pliku w logicznej strukturze systemu plików. Operacje wykonywane na plikach (takie jak odnajdywanie, odczyt czy usuwanie) są realizowane przy wykorzystaniu wpisów katalogów i metadanych. Dlatego organizacja struktury katalogów jest dla systemu bardzo istotna.

Najprostszym sposobem organizacji katalogu jest struktura jednopoziomowa (w praktyce niestosowana). W takiej strukturze informacje o wszystkich plikach ujęte są w jednym poziomie. Ma ona jednak istotne wady: każdy plik musi mieć globalnie unikatową nazwę, trudno oddzielać pliki różnych użytkowników, a wyszukiwanie staje się coraz mniej wygodne wraz ze wzrostem liczby plików.

Tych wad pozbawiona jest najczęściej wykorzystywana struktura katalogów zorganizowana w postaci drzewa.



Przykładowe drzewo katalogów

W katalogu drzewiastym drzewo ma swój katalog główny (ang. *root*), a każdy plik ma jednoznaczną ścieżkę określającą jego położenie w drzewie katalogów.

Format podanej ścieżki zależy od systemu operacyjnego. W wielu systemach (Unix, Linux, macOS) katalog jest reprezentowany przez ukośnik (ang. *slash* „/"). Przykładowa ścieżka wygląda następująco:

```
/home/czesio/podreczniki/podrecznik_syst_komputerowe/lekcja0/obrazki/edytp3.gif
```

W systemach Windows katalog jest reprezentowany przez odwrotny ukośnik (ang. *backslash* „\"), a pełna ścieżka zawiera też literę dysku (wolumenu). Przykładowo:

```
C:\Users\czesio\Documents\podreczniki\podrecznik_syst_komputerowe\lekcja0\obrazki\edytp3.gif
```

Oprócz pojęcia ścieżki bezwzględnej, w systemach opartych na drzewie istnieje również pojęcie ścieżki względnej – czyli drogi, jaką z aktualnego katalogu można dostać się do celu. W tym celu poruszamy się w górę drzewa do pierwszego katalogu wspólnego, korzystając ze specjalnego oznaczenia katalogu nadrzędnego: dwóch kropek („..").

Dysk logiczny

Wspomnieliśmy, że w systemach Windows przy podawaniu pełnej ścieżki często pojawia się litera dysku. W wielu systemach uniksowych „dyski logiczne" również istnieją, ale są widziane jako część jednego drzewa katalogów – po prostu montuje się je w wybranym miejscu (np. */home*, */mnt/dane*).

Definicja

Dysk logiczny, nazywany też partycją (ang. *partition*) lub wolumenem (ang. *volume*), jest wydzieloną **strefą** na fizycznym nośniku (albo logicznym urządzeniem zbudowanym z wielu nośników), na której znajduje się określony system plików.

Podziału na partycje dokonuje się w celu:

- uzyskania kilku rozłącznych obszarów na jednym dysku fizycznym;
- oddzielenia danych i systemu (np. osobny wolumen na dane użytkownika);
- budowania większych wolumenów z wielu nośników (np. macierze RAID, LVM, Storage Spaces).

> **Definicja**
>
> Dane umieszczone na jednym wolumenie są zorganizowane według reguł danego systemu plików i są niezależne od danych na innych wolumenach.

Niektóre operacje na plikach (np. zmiana nazwy czy przeniesienie pliku w obrębie jednego wolumenu) często nie wymagają kopiowania całej zawartości pliku – system plików może jedynie zmienić odpowiednie wpisy w metadanych. Dzięki temu takie operacje bywają bardzo szybkie.

Zabezpieczenia

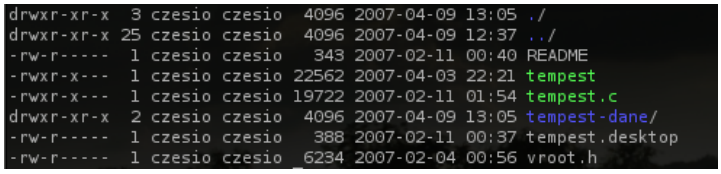
Najprostsze systemy plików nie uwzględniają ochrony plików i katalogów – każdy użytkownik może odczytywać i modyfikować wszystko. Podejście takie jest proste, ale w zastosowaniach innych niż bardzo proste środowiska (lub nośniki wymienne) jest mało rozsądne.

Po pierwsze, nie da się wtedy sensownie odseparować danych różnych użytkowników. Po drugie, łatwo doprowadzić do uszkodzenia systemu: jeśli każdy może modyfikować pliki systemowe, przypadkowe (lub złośliwe) działania mogą unieruchomić komputer.

Przykładem rodziny systemów plików o bardzo ograniczonych mechanizmach uprawnień są odmiany FAT, nadal spotykane na nośnikach wymiennych z powodu szerokiej kompatybilności.

W poważnych zastosowaniach stosuje się systemy plików umożliwiające kontrolę dostępu. Najczęściej spotyka się jeden z dwóch schematów:

- **klasyczne uprawnienia rwx** (Unix/Linux): każdy plik i katalog jest opisany przez właściciela i grupę oraz zestaw 9 podstawowych bitów uprawnień: odczyt/zapis/wykonanie dla właściciela, grupy i pozostałych;
- **listy kontroli dostępu (ACL)**: uprawnienia można nadawać bardziej szczegółowo, np. oddzielnie dla wielu użytkowników i grup, z większą liczbą typów uprawnień. ACL są powszechne w Windows (NTFS) i spotykane także w wielu systemach uniksowych.



Przykład uprawnień do plików w systemie Linux

Na rysunku widzicie przykładową listę uprawnień (to pierwsze 10 znaków). Litera *d* oznacza katalog, a następnie występują trzy grupy po trzy znaki: *rwx*. Jeśli dany znak nie występuje, to uprawnienie nie zostało przyznane. Dla pliku README właściciel ma prawo odczytu i zapisu, grupa ma prawo odczytu, a pozostali nie mają dostępu.

Księgowanie

Zabezpieczenia przed nieautoryzowanym dostępem nie gwarantują jeszcze bezpieczeństwa danych w sensie spójności po awarii. Bezpieczeństwo to także pewność, że w razie nagłej utraty zasilania lub zawieszenia się systemu struktura systemu plików nie zostanie „uszkodzona w połowie”.

Trzeba pamiętać, że zapis pliku to zwykle nie jest jedna, krótka operacja. System plików modyfikuje zarówno **dane** (zawartość pliku), jak i **metadane** (np. wpis katalogu, rozmiar pliku, mapę zajętości bloków). Dodatkowo część operacji może być buforowana w RAM i zapisywana na nośnik z opóźnieniem.

Wyobraźmy sobie prosty przykład: zapisujemy plik z danymi, np. „placa_2026.xlsx”. W praktyce system musi:

1. znaleźć wolne miejsce na nośniku (wolne bloki/klastry);
2. utworzyć/zmodyfikować metadane (np. dopisać wpis w katalogu, wskazać bloki, ustawić rozmiar pliku);
3. zapisać właściwe dane pliku (zawartość);
4. w razie potrzeby kontynuować alokację kolejnych bloków i uaktualniać metadane.

Jeśli awaria nastąpi pomiędzy zmianami metadanych a zapisem danych (albo w połowie zmian metadanych), system plików może pozostać w stanie niespójnym: na przykład pojawi się „plik-widmo” albo wycieknie wolne miejsce.

> **Definicja**

> Księgowanie (ang. *journaling*) to mechanizm, w którym system plików zapisuje informacje o planowanych zmianach w dzienniku (kronice), zanim zmodyfikuje właściwe struktury systemu plików. Po awarii system może na podstawie dziennika dokończyć operacje albo je wycofać tak, aby przywrócić spójność metadanych (a czasem także danych).

Po uruchomieniu systemu po awarii operacje w całości zapisane w dzienniku są „odtwarzane” (dokańczane), a operacje niekompletne są ignorowane. Dzięki temu czas przywracania spójności zwykle jest krótki i nie wymaga pełnego skanowania całego wolumenu.

W przypadku systemów plików bez księgowania, aby przywrócić porządek po nagłej awarii, trzeba uruchamiać narzędzia sprawdzające i naprawiające spójność struktur. Dla dużych wolumenów może to trwać długo.

Rozróżnia się dwa podstawowe podejścia do księgowania:

- **księgowanie metadanych** – zapewnia spójność struktur systemu plików, ale ostatnio zapisywane dane użytkownika mogą nie zostać w pełni zachowane;
- **księgowanie metadanych i danych** – dąży do zachowania także danych, kosztem dodatkowych operacji zapisu.

Komentarz

Istnieją też systemy plików oparte na innych technikach zwiększania spójności (np. mechanizmy typu copy-on-write). W tym podręczniku skupiam się jednak na klasycznym księgowaniu, bo najłatwiej na jego przykładzie zrozumieć, skąd biorą się problemy po awarii i jak je ograniczać.

1.2. Dostępne systemy plików

W praktyce spotyka się wiele systemów plików, a ich dobór zależy od zastosowania: inny system wybiera się dla nośnika wymiennego, inny dla dysku systemowego komputera, a jeszcze inny dla serwera czy pamięci masowej w centrum danych. Poniżej znajdziecie opis kilku popularnych (lub historycznie ważnych) rozwiązań.

FAT

System plików FAT (ang. *File Allocation Table*) wywodzi się z epoki MS-DOS i długo był podstawowym systemem plików w komputerach PC. Współcześnie nie jest natywnym systemem plików nowoczesnych systemów operacyjnych, ale nadal bywa spotykany na nośnikach wymiennych i w prostych urządzeniach (np. aparaty, rejestratory, firmware), głównie ze względu na szeroką kompatybilność.

W formacie FAT partycja (poza początkowymi zarezerwowanymi sektorami) jest podzielona na klastry (jednostki alokacji pliku). Każdy klaster składa się z jednego lub kilku sektorów i jest numerowany. System operacyjny na podstawie numeru klastra oblicza położenie danych na wolumenie.

Komentarz

Historycznie opisywano położenie danych przez geometrię dysku (CHS: ścieżka, głowica, sektor). Współcześnie system operacyjny i sterowniki używają adresowania logicznego (LBA), a szczegóły fizycznego rozmieszczenia danych są w dużej mierze ukryte przez kontroler dysku/SSD.

Klaster jest dla FAT jednostką niepodzielną – nie da się przydzielić „połowy klastra”. Plik zawsze zajmuje całkowitą liczbę klastrów, co przy dużych klastrach i wielu małych plikach może prowadzić do marnowania miejsca (tzw. strata na jednostce alokacji).

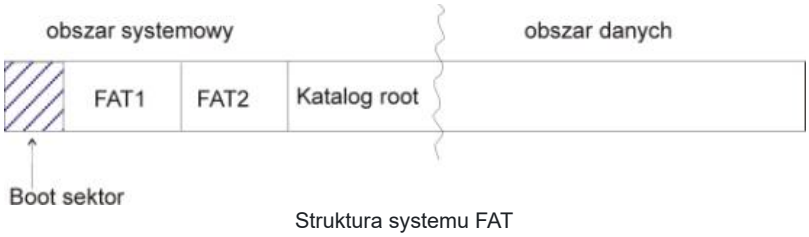
Pliki w systemie FAT organizowane są w katalogi. Każdy wpis katalogu zawiera nazwę pliku, atrybuty oraz numer pierwszego klastra. Lokalizacja dalszej części pliku jest wyznaczana przez tablicę alokacji plików.

Tablica alokacji plików (FAT) to struktura przechowywana na początku wolumenu. Zawiera ona tyle pozycji, ile jest klastrów. Każda pozycja opisuje stan klastra (np. wolny, zajęty, uszkodzony) albo wskazuje „następny klaster” danego pliku. W typowym opisie: wpis 0000h oznacza klaster wolny, a specjalne wartości oznaczają koniec łańcucha lub klaster uszkodzony.

Zaletą takiego podejścia jest prostota oraz łatwe „podążanie” po kolejnych fragmentach pliku. Wadą jest podatność na fragmentację (fragmenty pliku mogą leżeć daleko od siebie), a także koszt i ryzyko związane z częstą modyfikacją metadanych. Pewnym zabezpieczeniem w klasycznych odmianach FAT bywa przechowywanie dwóch kopii tablicy alokacji.

Dokładny podział wolumenu FAT wygląda następująco:

- obszar zastrzeżony (z boot sectorem) – zawiera m.in. BPB (BIOS Parameter Block) oraz kod startowy wolumenu;
- tablica (lub tablice) alokacji plików FAT;
- katalog główny i podkatalogi, zawierające nazwy plików, atrybuty, czasy utworzenia i modyfikacji oraz wskaźnik na pierwszy klaster danych;
- region danych – podzielony na klastry, zawierający właściwe dane plików.



Struktura systemu FAT

System plików FAT występuje w kilku wersjach, różniących się liczbą bitów używanych do adresowania klastrów.

FAT12

FAT12 to najstarsza wersja, kojarzona głównie z dyskietkami. Ze względu na bardzo małą liczbę możliwych do zaadresowania klastrów szybko stała się niewystarczająca i dziś ma znaczenie głównie historyczne.

Komentarz

> Wiele dawnych ograniczeń sprzętowych i programowych latami wpływało na rozwój komputerów, kompatybilność wsteczną i „dziwne” kompromisy w projektowaniu systemów. Część z nich w praktyce odczuwamy do dziś, choć w innej postaci (np. w ograniczeniach narzędzi lub formatów danych).

FAT16

FAT16 powstał jako rozwinięcie FAT12 i pozwalał obsłużyć większe wolumeny dzięki większej puli numerów klastrów. Wraz ze wzrostem pojemności dysków zwiększał się rozmiar klastra, co prowadziło do coraz większego marnowania miejsca przy małych plikach. Dodatkowo występowały ograniczenia rozmiaru partycji i kompatybilności z narzędziami systemowymi. Istotnym problemem historycznym były też krótkie nazwy plików (model 8.3) i ograniczenia wynikające ze sposobu kodowania znaków.

Wielkość klastra w systemie FAT16 zależy od wielkości partycji i jest ustalana podczas formatowania. Przykładowe wartości (historycznie spotykane) pokazuje tabela:

Wielkość partycji	**Liczba sektorów na klaster**	**Wielkość klastra**
0 MB–32 MB	1	512 bajtów
33 MB–64 MB	2	1 KB
65 MB–128 MB	4	2 KB
129 MB–255 MB	8	4 KB
256 MB–511 MB	16	8 KB
512 MB–1 GB	32	16 KB
1 GB–2 GB	64	32 KB

FAT32

W pewnym momencie ograniczenia FAT16 stały się zbyt uciążliwe i wprowadzono FAT32. Najważniejszy efekt praktyczny był taki, że można było obsługiwać znacznie większe wolumeny przy rozsądniejszych rozmiarach klastrów. W praktyce konkretne ograniczenia zależą od implementacji w systemie operacyjnym, narzędzi formatujących oraz rozmiaru klastra.

Najbardziej odczuwalne ograniczenie FAT32 w codziennych zastosowaniach to limit rozmiaru pojedynczego pliku: **4 GB minus 1 bajt**. To właśnie z tego powodu FAT32 bywa niewygodny dla dużych plików wideo, obrazów maszyn wirtualnych czy kopii zapasowych.

Przykładowe wielkości klastrów spotykane w FAT32 pokazuje tabela:

Wielkość partycji	**Wielkość klastra**
Poniżej 8 GB	4 KB
Od 8 GB do 16 GB	8 KB
Od 16 GB do 32 GB	16 KB
Powyżej 32 GB	32 KB

VFAT

Klasyczny FAT miał istotne ograniczenie nazewnictwa: nazwa pliku nie mogła mieć więcej niż 8 znaków plus 3 znaki rozszerzenia (format 8.3), a zestaw znaków był w praktyce ograniczony (ASCII i strony kodowe), co przez lata utrudniało wygodną pracę z narodowymi znakami diakrytycznymi.

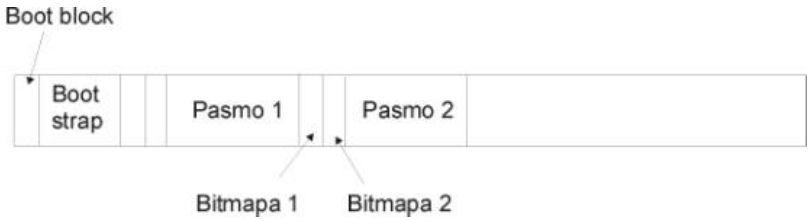
W starszych wersjach Windows wprowadzono obsługę długich nazw plików przez mechanizm VFAT – długie nazwy były przechowywane w specjalnych, dodatkowych wpisach katalogu. Rozwiązanie działa dobrze, ale trzeba pamiętać, że odczyt takiego wolumenu przez system lub narzędzie nieobsługujące VFAT może kończyć się widocznymi „krótkimi nazwami” 8.3 lub problemami interpretacji nazw.

FAT – podsumowanie

FAT jako system dysku systemowego to dziś w praktyce rozwiązanie przestarzałe: brakuje w nim natywnych mechanizmów uprawnień, kontroli dostępu i nowoczesnych zabezpieczeń spójności. Nadal ma jednak sens na nośnikach wymiennych i w urządzeniach, w których liczy się maksymalna kompatybilność.

W zastosowaniach domowych FAT32 jest często wypierany na nośnikach wymiennych przez **exFAT**, który lepiej radzi sobie z dużymi plikami i jest szeroko wspierany przez współczesne systemy.

HPFS



Struktura systemu HPFS

> ## Komentarz
>
> HPFS ma dziś przede wszystkim znaczenie historyczne (OS/2). Warto go jednak znać, bo pokazuje krok w stronę bardziej zaawansowanych struktur metadanych i wydajniejszego wyszukiwania.

System HPFS (ang. *High Performance File System*) był rozwijany głównie z myślą o OS/2. W porównaniu do FAT wykorzystywał bardziej złożone struktury do organizacji metadanych, m.in. rozwiązania oparte o drzewa (np. b-tree) dla katalogów. Taka organizacja przyspiesza wyszukiwanie plików, szczególnie gdy katalogów i plików jest dużo.

W porównaniu do FAT, HPFS był projektowany jako rozwiązanie bardziej „systemowe": z lepszą organizacją danych i z mechanizmami ważnymi w środowisku wieloużytkownikowym (m.in. kontrolą dostępu). Dla użytkownika przekładało się to na większy porządek i bezpieczeństwo pracy.

NTFS

NTFS (ang. *New Technology File System*) jest podstawowym systemem plików w rodzinie Windows NT (także we współczesnych wersjach Windows). Został zaprojektowany jako następca prostszych rozwiązań (np. FAT), z naciskiem na bezpieczeństwo, niezawodność i pracę w środowisku wieloużytkownikowym.

W porównaniu do FAT, NTFS oferuje m.in. księgowanie metadanych, listy kontroli dostępu (ACL) oraz bogatszy zestaw atrybutów plików. Teoretyczne limity rozmiaru pliku i wolumenu są bardzo duże, a praktyczne wartości zależą m.in. od rozmiaru klastra, wersji systemu i narzędzi administracyjnych.

Do często omawianych możliwości NTFS należą:

- mechanizm księgowania (dziennik zmian metadanych), zwiększający spójność po awarii zasilania;
- uprawnienia i ACL (precyzyjna kontrola dostępu dla użytkowników i grup);
- opcjonalne funkcje systemowe, takie jak szyfrowanie plików (EFS) czy kompresja – dostępne zależnie od edycji i konfiguracji systemu.

W NTFS metadane są przechowywane w specjalnych plikach systemowych. Jednym z kluczowych elementów jest Master File Table (MFT), zawierająca rekordy opisujące pliki i katalogi (ich atrybuty, rozmieszczenie danych itd.).

Warto też zauważyć, że w Windows wolumen NTFS może być „podpinany" nie tylko jako litera dysku, ale również jako punkt montowania w pustym katalogu (mechanizm znany od dawna w systemach uniksowych).

Extended File System – ext

W świecie Linuksa przez lata bardzo ważną rodziną systemów plików były ext2/ext3/ext4 (Extended File System). Rozwiązania te wywodzą się z klasycznych koncepcji uniksowych, w których kluczową rolę odgrywają struktury metadanych i-node oraz katalogi wiążące nazwy plików z numerami i-node.



Struktura systemu plików w systemie Unix

Na początku woluminu może znajdować się obszar rozruchowy. Ważnym elementem jest też **superblok** (superblock), zawierający informacje krytyczne dla działania systemu plików (np. rozmiary struktur, liczbę bloków, parametry wolumenu). Z racji znaczenia superbloku jego kopie bywają przechowywane w kilku miejscach, aby ułatwić odtwarzanie po awarii.

Za metadane plików odpowiadają **i-node'y**. Taki wpis przechowuje informacje o atrybutach (właściciel, grupa, uprawnienia), rozmiarze, czasach utworzenia i modyfikacji oraz wskaźniki do bloków danych. Nazwa pliku nie jest częścią i-node'a – nazwy są wiązane z numerami i-node w katalogach (katalog można traktować jako szczególny plik zawierający pary: nazwa → numer i-node).

Organizacja oparta o i-node'y zmniejsza wymagania pamięci operacyjnej: nie trzeba trzymać w RAM całej tablicy i-node'ów, a jedynie te, które są aktualnie potrzebne (np. dla otwartych plików). Zajętość pamięci zależy więc głównie od liczby otwartych plików i katalogów.

Drugą korzyścią z identyfikacji plików poprzez i-node'y jest możliwość stosowania **twardych odnośników** (hard links), czyli wielu nazw wskazujących na ten sam i-node. Dzięki temu ten sam plik może „występować" w kilku miejscach w drzewie katalogów. To także tłumaczy, dlaczego w systemach uniksowych usunięcie pliku (nazwy) nie musi oznaczać natychmiastowego usunięcia danych – dane znikają dopiero wtedy, gdy zniknie ostatnie odwołanie i żaden proces nie trzyma pliku otwartego.

> ## Komentarz
>
> Dzięki temu w systemie Linux możliwa jest aktualizacja oprogramowania bez przerywania jego pracy - nowe wersje plików są zapisywane, lecz stare są wciąż dostępne dopóki działa program. Jego nowa wersja będzie dostępna po ponownym uruchomieniu - jednak to może nastąpić już po zakończeniu procesu aktualizacji

Ext3 zastąpił ext2 głównie dzięki wprowadzeniu księgowania, a ext4 jest dziś powszechnie stosowany jako system domyślny w wielu dystrybucjach. W Linuksie spotyka się również inne systemy plików, np. XFS (często na serwerach) czy Btrfs (z dodatkowymi funkcjami zarządzania wolumenem). W niektórych środowiskach spotyka się także ZFS, choć nie jest on „domyślnym" wyborem w każdej dystrybucji.

Podsumowanie

Bez pamięci masowej trudno wyobrazić sobie pracę systemu komputerowego. Dane są przechowywane w postaci plików, a za ich organizację i udostępnianie odpowiada system plików. Od niego zależy sposób wykorzystania nośnika, bezpieczeństwo danych oraz wygoda pracy użytkownika.

W praktyce w Windows najczęściej spotyka się NTFS (a na nośnikach wymiennych także FAT32 lub exFAT). W systemach uniksowych i w Linuksie popularne są ext4 oraz inne systemy zależnie od potrzeb (np. XFS, Btrfs). Oprócz tego istnieją systemy przeznaczone do nośników optycznych (ISO 9660/CDFS, UDF) oraz systemy sieciowe i rozproszone (np. NFS, SMB). Nie wspomniałem tu szerzej o systemach klastrowych i rozproszonych, bo to temat na osobny rozdział.

Spis treści

1. System operacyjny - rozwiązania
1.1. Systemy operacyjne rodziny UNIX
1.2. Linux
1.3. Windows
1.4. macOS
1.5. Podsumowanie

1. System operacyjny - rozwiązania

W trakcie tej lekcji chcemy Wam przedstawić systemy operacyjne od strony użytkowej. Zapoznamy Was z historią rozwoju systemów operacyjnych, jak również poruszymy zagadnienia ochrony i bezpieczeństwa systemów.

Lekcja ta zawiera krótki i ogólny opis kilku wybranych systemów operacyjnych oraz ich porównanie. Dzięki niej uzyskacie szerszą perspektywę na dostępne na rynku systemy operacyjne oraz ich najważniejsze cechy.

Historia rozwoju systemów operacyjnych

Systemy operacyjne w czasie swego istnienia przeszły znaczące zmiany. Pierwsze z nich były bardzo proste i mało wydajne. Dzisiejsze systemy operacyjne są systemami o dużym stopniu skomplikowania, zajmują gigabajty pamięci, a ich możliwości są niewyobrażalna w porównaniu z możliwościami ich odpowiedników sprzed lat.

Do historycznych już dzisiaj rozwiązań w dziedzinie systemów komputerowych możemy zaliczyć wymienione w lekcji 5 systemy wsadowe czy systemy wieloprogramowe. Ich stosowanie wymagało przede wszystkim wiedzy ... głębokiej wiedzy informatycznej i znajomości budowy komputera.

Reprezentantem tego typu rozwiązań był m.in. MS-DOS. Był to system operacyjny o bardzo ograniczonym modelu zarządzania zasobami: typowo jednoużytkownikowy, bez ochrony pamięci i bez preempcji w sensie znanym ze współczesnych systemów. Z tego powodu nie będziemy się nim dalej zajmowali, traktując go jako ważny etap historyczny, ale nie jako punkt odniesienia dla dzisiejszych mechanizmów bezpieczeństwa i wielozadaniowości.

Innym historycznym systemem o zbliżonej do MS-DOS funkcjonalności był CP/M - przeznaczony dla 8-bitowych komputerów.

W latach sześćdziesiątych wśród producentów komputerów pojawił się trend ujednolicania produkowanych przez nich linii sprzętu. Dotąd bowiem jednostki z różnych linii (mainframe, komputery do prostych zastosowań) były wzajemnie niekompatybilne, czyli niemożliwym było uruchomienie na nich tego samego programu bez zmiany jego struktury (w zasadzie napisania go od nowa). Z racji niewygody i dużych kosztów utrzymania tego typu rozwiązania, najpierw firma IBM, a za nią kolejni producenci zdecydowali się na ujednolicenie architekturalne swoich produktów. W ten sposób powstał IBM System/360, seria komputerów różniących się wydajnością i ceną ale tej samej architekturze. Komputery tej serii obsługiwane były przez system operacyjny OS/360. Najważniejszą cechą tego systemu było wprowadzenie możliwości wykonania (części lub całego) innego programu w czasie, kiedy bieżący program oczekiwał na zakończenie operacji wejścia wyjścia (np. pobrania danych z taśmy) zwane multiprogrammingiem. Oczywiście system operacyjny musiał zarządzać dostępem do komponentów komputera (szczególnie pamięci) aby wiele uruchomionych jednocześnie programów nie nadpisywało wzajemnie swoich danych. Zarządzanie to było wspomagane przez odpowiednie rozwiązania sprzętowe wprowadzone w serii System/360.

Sukces OS/360 i ogromne zainteresowanie multiprogrammingiem zapoczątkowały prace nad systemami operacyjnymi umożliwiającymi wielodostępność (time-sharing) oraz sprawny podział czasu procesora pomiędzy wielu użytkowników i zadania. Zaowocowało to projektem systemu CTSS na uniwersytecie M.I.T, a następnie szeroko zakrojonym projektem MULTICS (1969 r.) realizowanym przez MIT, GE i Bell Labs. System ten był projektowany w celu umożliwienia korzystania z jednego komputera typu mainframe (GE 645) tysiącom użytkowników jednocześnie (w oryginalnym projekcie docelową grupą byli mieszkańcy miasta Boston). Projekt ten nie odniósł sukcesu komercyjnego, mimo to system był wykorzystywany przez takie jednostki jak NSA (Agencja Bezpieczeństwa Narodowego USA), firmy Ford i GM, Air Force Data Services Center w USA, Industrial Nucleonics oraz Kanadyjski Departament Obrony Narodowej działający do 30 października 2000. System ten posiadał wiele rozwiązań, które zostały wykorzystane w systemach operacyjnych następnej generacji i wykorzystywane są do dzisiaj (np. koncepcje ochrony pamięci, kontroli dostępu i hierarchicznych systemów plików).

Jednakże zarówno OS/360, jak MULTICS nie zdobyły masowej popularności wśród użytkowników komputerów osobistych - natomiast doświadczenia zdobyte w tych projektach okazały się bezcenne. Twórcy MULTICS postanowili wykorzystać je przy opracowaniu systemu operacyjnego dla prostszego i tańszego (jak na ówczesne czasy) komputera DEC PDP-7 - i tu przechodzimy do rodziny rozwiązań, które w różnych odmianach przetrwały do dziś, czyli systemów uniksowych (Unix i Unix-like).

Rodzina Unix

Pierwsza wersja systemu UNIX powstała w Bell Labs firmy AT&T w stanie New Jersey w 1969 na komputery architektury PDP-7 i PDP-9 firmy DEC. Jej głównymi autorami byli Ken Thompson i Dennis Ritchie. Początkowy cel stworzenia nowego systemu operacyjnego został osiągnięty i autorzy rozpoczęli prace nad drugą wersją systemu na komputery PDP-11. Tę wersję uruchomiono w 1971 roku. W tym czasie ukazał się Unix Programmer's Manual, który był de facto opisem ówczesnego Uniksa. Drugie wydanie systemu Unix pojawiło się w 1972 roku, w którym wprowadzono pojęcie łącza komunikacyjnego łączącego procesy. Minikomputery PDP-11 uzyskały wielką popularność i wraz z nimi także Unix. Dlatego wkrótce napisano trzecią wersję systemu operacyjnego Unix.

Autorzy zrezygnowali z pisania systemu w asemblerze i przenieśli całość kodu do stworzonego przez Dennisa Ritchie wysokopoziomowego języka C, co zaowocowało przenośnością systemu, oraz - paradoksalnie - popularyzacją języków wysokiego poziomu w programowaniu systemowym. Język C do dziś pozostaje jednym z najważniejszych języków systemowych.

W 1975 roku pojawił się Unix Szóstego Wydania - inaczej Wersja Szósta - rozprowadzana nieodpłatnie w uczelniach dla zastosowań akademickich. Przyczyniło się to do gwałtownego rozwoju systemu, powstawania poza AT&T rozszerzeń i oprogramowania. Legendą tego okresu stała się napisana przez profesora Uniwersytetu Nowej Południowej Walii w Australii, Johna Lionsa, tzw. Lions Book zawierająca pełny

kod V.6 wraz z komentarzem wers po wersie.

W 1978 ukazała się wersja 7 (Version 7) - była pierwszą szeroko dostępną, choć AT&T nie zapewniał obsługi i serwisu. Jej zestaw funkcji i programów był, aż do czasu ukazania się specyfikacji IEEE i POSIX, główną bazą unifikującą rodzące się różne odmiany Uniksa. Jednym z ważniejszych wydarzeń we wczesnej historii systemu Unix był projekt agencji DARPA przyjęcia jednego systemu operacyjnego na wszystkich komputerach ARPANET. DARPA zdecydowała, że tym systemem będzie wersja Berkeley systemu Unix. W ramach pierwszego kontraktu, który trwał w latach 1981-1982, powstał system 4BSD oraz 4.1BSD. Sukces zaowocował kolejnym dwuletnim kontraktem, którego owocem był 4.2BSD, będący pierwszym systemem operacyjnym zawierającym obsługę TCP/IP.

Wraz z początkiem lat 80 AT&T zdecydowała się na komercjalizację Uniksa. W 1980 roku licencję na rozwijanie jego kodu sprzedano firmie Santa Cruz Operations, która na zlecenie Microsoftu, miała przygotować Xenix, jego wersję Uniksa. W 1983 roku ukazała się pierwsza wersja komercyjna Uniksa z AT&T, System V.

Wraz z komercjalizacją AT&T przestała udostępniać kod źródłowy systemu poza licencjami komercyjnymi. Spowodowało to protesty wśród wielu inżynierów akademickich (UCB, MIT), którzy do tej pory pisali własne rozszerzenia systemu i uczestniczyli w jego rozwoju (vide: BSD). Blokada nałożona na kod stała się jedną z przyczyn rozwoju ruchu wolnego oprogramowania. Założona w 1983 roku przez Richarda Stallmana z MIT Free Software Foundation postawiła sobie za cel stworzenie wolnego systemu uniksowego bez kodu pochodzącego z AT&T.

Popularność systemu i mnogość jego odmian wymusiła potrzebę standaryzacji API systemowego, który każdy z systemów deklarowany jako „kompatybilny z Unixem" musiał obsługiwać. Standard ten, wprowadzony przez IEEE nosi oficjalną nazwę IEEE 1003, a w ISO/IEC jest publikowany jako ISO/IEC 9945; w świecie komputerów jest on bardziej znany pod nazwą **POSIX**. W praktyce, we współczesnych systemach spotkacie zarówno systemy certyfikowane (np. w ramach Single UNIX Specification), jak i systemy bardzo zgodne, ale bez formalnej certyfikacji. patrz: IEEE 1003.1 (POSIX.1), patrz: ISO/IEC 9945

Ze standardem POSIX w dużym stopniu zgodna jest większość współcześnie spotykanych systemów typu Unix-like, w tym Linux oraz systemy z rodziny BSD (np. FreeBSD). Systemy Apple (macOS i iOS) wywodzą się z tradycji uniksowej i udostępniają szeroką część API w stylu POSIX. Z kolei Android, choć używa jądra Linux i posiada wiele podobnych mechanizmów, nie jest pełnym „systemem POSIX" w sensie formalnym i kompatybilność zależy od konkretnego interfejsu i warstwy programistycznej.

Wśród historycznych systemów uniksowych, które w swoim czasie miały duże znaczenie w zastosowaniach serwerowych i specjalistycznych, były m.in. AIX, HP-UX czy Solaris, a także rozwiązania dziś już nieobecne na rynku masowym (np. IRIX, UnixWare). Warto zauważyć, że właśnie ta różnorodność była jednym z powodów, dla których standaryzacja interfejsów (POSIX) stała się tak ważna.

Więcej informacji o historii i rozwoju Unix-a możecie znaleźć między innymi pod adresem http://www.levenez.com/unix

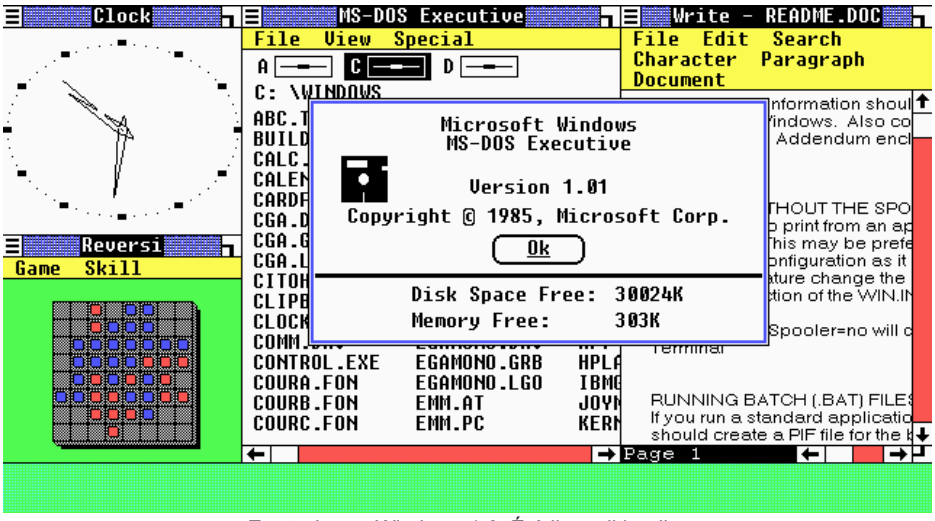
Rodzina Windows

Historia rozwoju systemów Windows dzieli się na dwie odnogi. Punktem startowym dla obu z nich był MS-DOS, opracowany i wypuszczony w 1981 roku.

Ze względu na fakt, iż nie był to system operacyjny sensu stricte w rozumieniu dzisiejszych mechanizmów ochrony i wielozadaniowości, w zasadzie od samego początku istnienia IBM PC (a szczególnie od pojawienia się modelu PC AT) był poszukiwany dla niego następca, tym bardziej, że niedługo później pojawiły się rozwiązania systemów operacyjnych dla innych platform (takich jak MacOS firmy Apple, Kickstart+Workbench przeznaczony dla Amigi czy TOS stosowany w Atari ST) bijące na głowę funkcjonalnością i jakością MS-DOS.

Firma Microsoft zdecydowała się na poszukiwanie tego rozwiązania dwutorowo - z jednej strony zajęła się rozbudową nakładek na DOS, które potem przekształciły się w tzw. domowe wersje Windows, druga - rozpoczęta we współpracy z IBM miała doprowadzić do zbudowania zupełnie nowego systemu operacyjnego. Ten swoisty dualizm skutkował pojawianiem się na rynku systemów domowych (o znacznie bardziej chwytliwej dla oka stylistyce, lecz ograniczonych w kwestiach pracy w sieci i bezpieczeństwa) oraz wersji profesjonalnych (uboższych graficznie, lecz stabilniejszych podczas pracy w sieci i wyposażonych w lepsze zabezpieczenia).

Pierwsza niezależna wersja Microsoft Windows (opatrzona numerem 1.0), obecna na rynku od listopada 1985, nie oferowała dużego zakresu funkcjonalności i nie zyskała szerszego uznania. Windows 1.0 nie był systemem operacyjnym, lecz przedłużeniem - nakładką na system MS-DOS (jak praktycznie wszystkie wersje domowe tego systemu).



Zrzut ekranu Windows 1.0. Źródło - wikipedia.

Microsoft Windows w wersji 2.0, sprzedawany od listopada 1987, nieco lepiej przyjął się wśród użytkowników. Wersja 2.03, datowana na styczeń 1988, wprowadziła zmiany w wyglądzie - zamiast okien ułożonych sąsiadująco możliwe było nachodzenie poszczególnych aplikacji na siebie. Skutkiem wprowadzenia tej zmiany był pozew wytoczony Microsoftowi przez firmę Apple Computer, który zarzucał firmie Billa Gatesa naruszenie praw autorskich należących do Apple. Wersja 3.0 nakładki Windows, sprzedawana od roku 1990, była pierwszym produktem rodziny, który osiągnął szerszy sukces. Wyznacznikiem popularności było sprzedanie 2 milionów kopii nakładki w trakcie pierwszych sześciu miesięcy obecności rynkowej. Wprowadzone ulepszenia dotyczyły przede wszystkim interfejsu użytkownika i możliwości pracy wielozadaniowej. W sierpniu 1995 Microsoft rozpoczął sprzedaż systemu operacyjnego Windows 95, który kontynuował zmiany w interfejsie użytkownika i wprowadzał preempcję dla 32-bitowych aplikacji, przy jednoczesnym zachowaniu kooperatywnego modelu dla starszych aplikacji 16-bitowych. Kolejnym systemem tej odnogi rodziny była wersja Windows 98, który na rynku pojawił się w czerwcu 1998. Dość ostrej krytyce poddana została jego powolność w porównaniu z Windows 95, lecz większość niedogodności została wyeliminowana w wydanej w roku 1999 wersji Windows 98 Second Edition. Ostatnim przedstawicielem tej gałęzi było Windows ME.

Niezależnie od wprowadzanych fajerwerków graficznych, systemy Windows z serii domowej ciągle w dużej mierze opierały się o jądro MS-DOS ze wszystkimi jego ograniczeniami (między innymi z kooperatywną wielozadaniowością bez wywłaszczania), nie miały możliwości ochrony pamięci w sensie współczesnym, nie pozwalały na sensowne zabezpieczenie komputera i często były niestabilne. Dlatego też ich rozwój został zarzucony, wycofano je ze sprzedaży, a rolę systemów domowych przejęły kolejne wersje linii NT (początkowo sprzedawane w różnych edycjach, a z czasem rozwijane jako jedna główna linia systemu).

Z drugiej strony, w 1987 roku światło dzienne ujrzała pierwsza wersja wspólnego dzieła IBM i Microsoftu - OS/2 1.0. Był to wtedy system z tekstową powłoką, ale posiadający już bogate API zapewniające kontrolę grafiki i obsługę myszy. Interfejs graficzny wprowadzono wraz z wydaniem w listopadzie roku 1988 wersji 1.1.

W roku 1990 współpraca firm Microsoft i IBM nad OS/2 rozluźniła się. Rosnąca popularność środowiska graficznego Windows opartego o DOS skłoniła Microsoft do skupienia uwagi na własnym systemie. Doprowadziło to do podziału prac nad OS/2 – IBM miał zająć się wydaniem wersji 2.0, podczas gdy Microsoft miał skupić się nad wersją 3.0 znaną jako NT OS/2. Ostatecznie współpraca obu firm nad OS/2 została zerwana, zaś wersję 3.0 Microsoft wydał jako Windows NT.

Komentarz

Wydana samodzielnie przez IBM w roku 1992 wersja 2.0 systemu OS/2 zawierała nowy interfejs graficzny oraz 32-bitowe API. Prawdziwym 32-bitowym systemem był jednak dopiero OS/2 w wersji 3 wydanej w roku 1994. Wprowadzał on ponadto lepszą obsługę multimediów i internetu. Potem pojawił się system OS/2 Warp, który w swoim czasie był rozwijany jako nowoczesna alternatywa dla systemów z rodziny DOS/Windows. Niestety - IBM wycofał OS/2 ze sprzedaży 23 grudnia 2005 roku, a 31 grudnia 2006 roku zakończył wsparcie dla tego systemu.

W lipcu 1993 pojawił się Windows NT, oparty na nowym jądrze. Seria NT była postrzegana jako system dla profesjonalistów. Następnym przedstawicielem tej serii był system operacyjny Windows 2000. W październiku 2001 Microsoft przedstawił XP. Produkt ten łączył w sobie jądro przejęte z linii Windows NT oraz funkcjonalność systemu Windows 95 i jego następców. W prasie komputerowej pojawiło się sporo pochwał skierowanych pod adresem nowego systemu (pomijając jego szatę graficzną). Oferowane były dwie wersje Windows XP, ochrzczone mianami "Home" i "Professional", przy czym pierwsza - skierowana do użytkowników domowych - nie była wyposażona w szereg mechanizmów sieciowych i zabezpieczających, które zostały wprowadzone w wersji Professional. Dodatkowym produktem linii XP była wersja "Media Center", która pojawiła się w roku 2003 i skierowana była głównie do entuzjastów filmów telewizyjnych i DVD, pozwalając na przykład na nagrywanie sygnału telewizyjnego i obsługę za pomocą pilota. W 2007 roku pojawiła się wersja Windows Vista, tym razem już w 6 odmianach, lecz wciąż oparta na jądrze NT, potem Windows 7 i Windows 8.

Na tym samym jądrze NT oparte są również edycje Windows przeznaczone do zastosowań serwerowych. Wprowadzały one wiele modyfikacji w stosunku do wersji przeznaczonej dla stacji roboczych, głównie w dziedzinie bezpieczeństwa i centralnego zarządzania (domeny, Active Directory, itd.). Historycznie ważnymi wydaniami były m.in. Windows Server 2003 oraz Windows Server 2008. W kolejnych latach pojawiły się Windows Server 2012 i 2012 R2, a następnie nowsze wydania linii serwerowej, takie jak Windows Server 2016, 2019, 2022 oraz nowsze wersje z bieżącego cyklu rozwoju. Jeśli chcecie sprawdzić aktualną, oficjalną informację o obecnie wspieranych wydaniach i kanałach wydań, zajrzyjcie do dokumentacji Microsoftu. patrz: Windows Server release information (Microsoft Learn)

Systemy chmurowe i ich historia

Gdy mówimy o systemach operacyjnych, łatwo skupić się na „systemie zainstalowanym na jednym komputerze". Zauważcie jednak, że we współczesnych zastosowaniach coraz częściej „systemem" staje się cały ekosystem: warstwa wirtualizacji, zarządzanie zasobami centrum danych, sieć programowalna i mechanizmy bezpieczeństwa. W takim ujęciu kluczową rolę pełnią systemy i platformy chmurowe, które automatyzują uruchamianie usług na setkach lub tysiącach maszyn.

Historycznie idee, które doprowadziły do chmury, mają wspólne korzenie z time-sharingiem (wielodostępnością) znanym już z CTSS czy MULTICS: wielu użytkowników korzysta ze współdzielonych zasobów, a system rozdziela je dynamicznie. Różnica polega na skali i sposobie dostarczania usług: chmura opiera się na wirtualizacji, automatyzacji i rozliczaniu „na żądanie". patrz: History of cloud computing

Za jeden z przełomów nowoczesnej chmury często uznaje się start usług infrastrukturalnych (IaaS) w Amazon Web Services: S3 oraz EC2 zostały uruchomione w 2006 roku i spopularyzowały model wynajmowania mocy obliczeniowej jako usługi. patrz: History of cloud computing (AWS S3/EC2)

W kolejnych latach pojawiły się następne duże platformy publiczne, np. Microsoft Azure (uruchomiony w 2010 roku). Równolegle zaczęły dojrzewać platformy do budowy chmur prywatnych i hybrydowych, w tym OpenStack (2010) jako projekt open-source umożliwiający budowę własnej infrastruktury chmurowej na standardowym sprzęcie. patrz: History of cloud computing (Azure, OpenStack)

Warto też zauważyć drugi trend: odejście od myślenia „maszyna wirtualna = jednostka wdrożenia" w stronę kontenerów i orkiestracji. W praktyce oznacza to, że coraz częściej uruchamiamy aplikacje jako zestaw usług w kontenerach, a system zarządzania klastrem dba o skalowanie, odtwarzanie po awarii, rozmieszczenie na węzłach i aktualizacje. Najbardziej znanym przedstawicielem tej klasy jest Kubernetes, udostępniony jako open source w 2014 roku i inspirowany wcześniejszymi doświadczeniami Google w zarządzaniu klastrami. patrz: Kubernetes origin story (Google Cloud Blog), patrz: Borg, Omega, and Kubernetes (publikacja)

Jeśli więc dziś pytacie „jaki system operacyjny działa w chmurze?", odpowiedź bywa wielowarstwowa: na węzłach nadal zwykle pracuje Linux lub Windows Server, ale ponad nimi działa dodatkowa warstwa zarządzania (platforma chmurowa i/lub orkiestrator), która w dużym stopniu przejmuje rolę klasycznego „systemu operacyjnego" w skali centrum danych.

Podsumowanie

W dzisiejszych czasach wykorzystywany system operacyjny zależy od zastosowania systemu komputerowego, z którym jest związany. Aby sprostać odmiennym wymaganiom stawianym systemom w każdej grupie zastosowań, rozwinęły się one w kilku różnych kierunkach.

W komputerach osobistych zastosowanie mają głównie wieloprogramowe systemy operacyjne z podziałem czasu (np. Microsoft Windows, macOS, Linux). W systemach tych główny nacisk nie został położony na maksymalizację wydajności i wykorzystania procesora, a na wygodę użycia i interakcję systemu z użytkownikiem. Systemy te posiadają przyjazny dla człowieka graficzny interfejs.

W zastosowaniach serwerowych (maszyny obliczeniowe, serwery baz danych itp.) wykorzystuje się głównie skomplikowane, wieloprogramowe, równoległe systemy z podziałem czasu lub systemy rozproszone. W systemach tych nacisk położony jest na szybkie i bezawaryjne wykonywanie zadań, do których dany system jest przeznaczony. W tym celu zwykle stosuje się w jednym systemie wiele procesorów oraz dubluje się np. zasoby dyskowe. Interfejs użytkownika w takich systemach ma mniejsze znaczenie. We współczesnych wdrożeniach bardzo często spotkacie też model chmury i rozwiązań hybrydowych, gdzie ponad klasycznym systemem operacyjnym działa warstwa platformy chmurowej oraz automatyzacji.

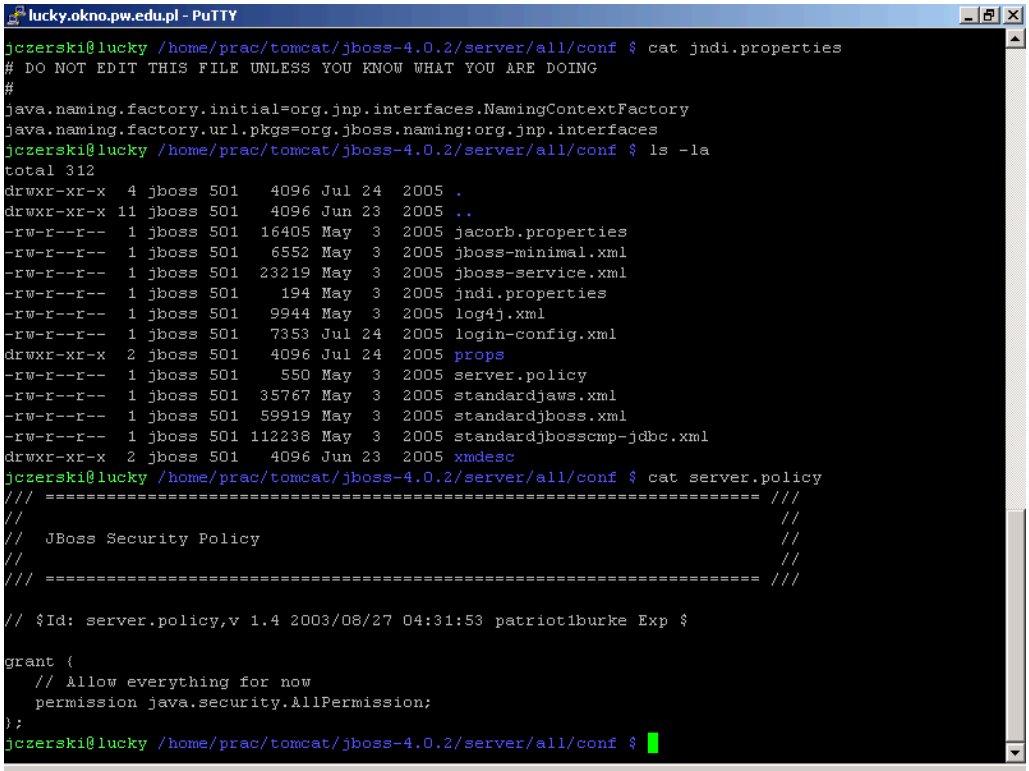
1.1. Systemy operacyjne rodziny UNIX

Pierwsza wersja systemu UNIX została opracowana w 1969 roku przez zespół pod nadzorem Kena Thomsona. Do czasów dzisiejszych Unix doczekał się wielu kolejnych, coraz bardziej udoskonalonych wersji (zarys historii systemu zamieściliśmy w pierwszym segmencie tej lekcji). Zazwyczaj stosuje się go na komputerach „większych" niż komputery domowego użytku - stanowił on podstawę wielu klasycznych serwerów i stacji roboczych. W praktyce współczesne klastry obliczeniowe i superkomputery są niemal w całości oparte o systemy Linux (rodzina Unix-like). patrz: TOP500 - Operating system family

W zastosowaniach domowo-biurowych oraz jako typowe stacje robocze, „czyste" komercyjne Uniksy (np. Solaris) stały się współcześnie rozwiązaniami niszowymi, a niektóre historyczne systemy (np. IRIX) przeszły do historii. Ich miejsce w praktyce zajęły przede wszystkim Windows oraz rodzina Unix-like: Linux i systemy BSD, a także systemy Apple (macOS) wywodzące się z tradycji uniksowej.

Unix został zaprojektowany jako system z podziałem czasu. Cechują go także wielodostępowość i wielozadaniowość. W nowszych wersjach systemów uniksowych zastosowano stronicowanie na żądanie jako mechanizm wspierający zarządzanie pamięcią i podejmowanie decyzji o przydziale procesora. Rodzaj schedulera, techniki zarządzania pamięcią czy same systemy plików mogą się różnić między wersjami Unix-ów - bo też są ich różne zastosowania (np. jako serwery, systemy czasu rzeczywistego, itp...)

Standardowym interfejsem użytkownika jest interpreter poleceń tzw. powłoka (ang. *shell*). Istnieje kilka odmian shell'a; jedną z najczęściej spotykanych jest Bash (ang. *Bourne-Again Shell*), ale w niektórych systemach domyślną powłoką bywa inna (np. zsh). Sposób wydawania poleceń w Unix ma charakter tekstowy, choć pod kontrolą tego systemu mogą działać systemy graficzne (np. X Window System).



Przykładowy wygląd ekranu shell'a w czasie pracy w systemie typu Unix.

Architektura systemu

System UNIX zbudowano trzymając się kilku ważnych zasad. Główną zasadą jest tak zwana reguła KISS (ang. *Keep It Simple, Stupid* – tłum: „to ma być proste, głupku"). Interpretacja tej reguły przy tworzeniu systemu Unix zaowocowała prostotą i przejrzystością architektury i budowy systemu. Trzymano się zasady, że każdy program ma wykonywać jedną i tylko jedną rzecz, do której został przeznaczony i ma ją wykonywać dobrze.

Do wykonywania bardziej skomplikowanych operacji można programy łączyć w tzw potoki - wtedy to wyjście z jednego z nich staje się wejściem do drugiego, ponadto wykonają się automatycznie jeden po drugim. Przykładowo:

```
ls -la | grep mop
```

Wyświetli nam wszystkie nazwy plików i podkatalogów, które zawierają następujące po sobie litery mop. Do wykonania tego zadania wykorzystaliśmy dwa programy - **ls** zwraca nam listę plików i podkatalogów, przełącznik *-la* oznacza że polecenie **ls** ma pokazać detale plików. Potem napotykamy pionową kreskę - **|** która jest jednym z symboli łączenia w potok i oznacza mniej więcej: weź wynik i przekaż następnemu poleceniu jako wejście. Następne polecenie to **grep** - filtr wyrażeń regularnych, przy wywołaniu jak powyżej po prostu "przepuści" (wyświetli)

na ekranie tylko linie zawierające litery mop. Jeśli chcielibyśmy wyszukiwać linie z literami mop nie w katalogach, lecz w pliku dajmy na to 'eleonora.txt', wywołanie zmieniłoby się tylko w pierwszej części - dotyczącej dostarczenia danych do wyszukiwania (zamiast ls w tym wypadku wykorzystamy plik):

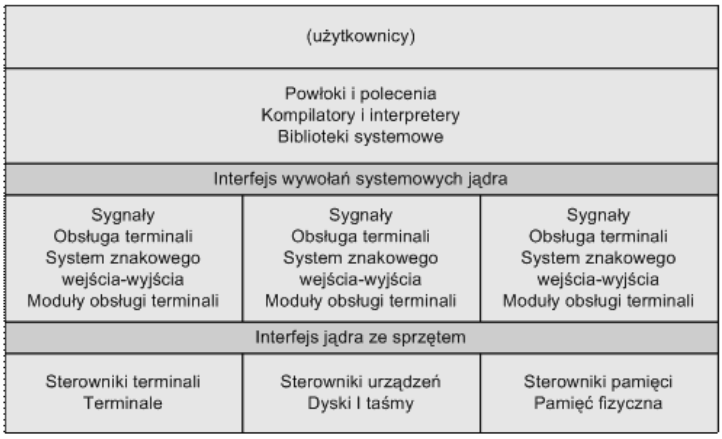
```
cat eleonora.txt | grep mop
```

Stosując przetwarzanie potokowe poleceń, w połączeniu z dobrze dobranym zestawem najprostszych programów wykonujących podstawowe operacje, można wykonać dowolne, nawet najbardziej skomplikowane czynności. Drugim bardzo ważnym elementem konstrukcji systemu jest reguła „wszystko jest plikiem”. Oznacza ona, że zarówno pliki na dysku, partycje, całe dyski, nośniki taśmowe porty komputera czy urządzenia wejścia-wyjścia takie jak np. mysz są traktowane jako plik i istnieją w strukturze systemu plików. Szczegóły związane z obsługą urządzeń zawarte są w jądrze systemu lub osobnych modułach do tego przeznaczonych dołączanych dynamicznie do jądra. Sam system plików jest zbudowany na zasadzie wielopoziomowego drzewa. W doborze większości algorytmów kierowano się prostotą, a nie szybkością lub wyrafinowaniem.

Większość kodu systemu została napisana w języku C, który rozwijano w Bell Labs m.in. na potrzeby programowania systemowego i przenośności kodu. Dzięki zastosowaniu języka wyższego poziomu (wcześniej stosowano Asembler), łatwiejsze stało się przenoszenie oprogramowania uniksowego z jednego systemu sprzętowego na inne.

W początkowym okresie rozwoju Uniksa kod źródłowy bywał udostępniany (szczególnie w środowiskach akademickich), co ułatwiało analizę i rozwój systemu, ale nie oznaczało jeszcze „open source” w dzisiejszym sensie. Dopiero później rozwinęły się ruch wolnego oprogramowania i otwarte licencje, które umożliwiły powstawanie w pełni otwartych systemów uniksowych i unix-like (np. rodziny BSD oraz Linux).

Podobnie jak większość systemów operacyjnych, Unix składa się z dwu oddzielnych części: jądra systemu i programów systemowych. Na system operacyjny można patrzeć jak na system złożony z warstw:



Warstwowa struktura systemu typu Unix..

Wszystko to, co znajduje się poniżej interfejsu funkcji systemowych i powyżej sprzętu fizycznego, tworzy jądro. W jądrze zrealizowany jest system plików, planowanie przydziału procesora, zarządzanie pamięcią i inne funkcje systemu operacyjnego, do których dostęp odbywa się za pomocą wywołań systemowych. Odwołania do systemu Unix można z grubsza podzielić na trzy kategorie: manipulowanie plikami, sterowanie procesami oraz manipulowanie informacją.

System plików

Plik w systemie Unix jest ciągiem bajtów. Jądro nie narzuca plikom żadnej struktury. Pliki są zorganizowane w drzewiastą strukturę katalogów. Katalogi też są plikami, które zawierają informacje o tym, jak odnaleźć inne pliki. Nazwą ścieżki do pliku jest napis, który identyfikuje plik przez określenie drogi prowadzącej do tego pliku w strukturze katalogowej.

Pod względem składni nazwę ścieżki tworzą poszczególne nazwy plików pooddzielane ukośnymi kreskami. Na przykład w `/usr/local/font` pierwsza kreska symbolizuje korzeń drzewa katalogów, zwany katalogiem głównym (ang. *root*). W przeciwieństwie do systemów MS Windows, gdzie każdy dysk sformatowany jest w jakimś systemie plików i stanowi oddzielny korzeń dla struktury plików, w systemie plików Unix’a katalog główny może obejmować wiele dysków czy partycji. Następny element ścieżki `usr` oznacza podkatalog korzenia, `local` oznacza podkatalog katalogu usr, a `font` jest nazwą pliku lub katalogu docelowego. Na podstawie składni nazwy ścieżki nie można wywnioskować, czy nazwa font odnosi się do pliku zwykłego czy do katalogu.

W systemie Unix używa się zarówno bezwzględnych (rozpoczynających się od korzenia, czyli ukośnej kreski), jak i względnych (rozpoczynających się w katalogu bieżącym) nazw ścieżek. Plik może być znany pod więcej niż jedną nazwą w jednym lub większej liczbie katalogów. Takie zwielokrotnione nazwy są nazywane dowiązaniami (ang. *link*), przy czym dowiązania twarde wskazują na ten sam obiekt systemu plików (ten sam i-węzeł), a dowiązania symboliczne (ang. *symbolic link*) są oddzielnymi plikami, które przechowują ścieżkę do innego obiektu. Te dwa rodzaje dowiązań znane są również jako dowiązania twarde i miękkie. W odróżnieniu od dowiązania twardego, dowiązanie miękkie może wskazywać katalog, może „wisieć” (gdy cel zostanie usunięty) i może wykraczać poza granice pojedynczego systemu plików. Występująca w katalogu nazwa „.” jest twardym dowiązaniem do tego samego katalogu, nazwa „..” oznacza twarde dowiązanie do katalogu nadrzędnego.

Bezpieczeństwo

Systemy rodziny Unix uznawane są jako systemy z dobrze zaprojektowaną i zaimplementowaną polityką bezpieczeństwa. Zapewnienie bezpieczeństwa danych w systemie wielodostępowym, jakim jest Unix, sprowadza się do rozwiązania dwóch kwestii: uwierzytelniania użytkowników oraz kontroli dostępu do zasobów.

Współcześnie uwierzytelnianie hasłem w systemach Unix-like jest zwykle rozdzielone: plik z informacjami o kontach użytkowników (np. `/etc/passwd`) bywa udostępniany do odczytu, natomiast zaszyfrowane (a ściślej: zahaszowane) reprezentacje haseł i polityki haseł przechowuje się w plikach o ograniczonych uprawnieniach (np. `/etc/shadow` w Linuksie). Hasło użytkownika jest łączone z wartością losową (solą), a wynik jest przetwarzany funkcją jednokierunkową; nie oznacza to, że „złamanie" jest absolutnie niemożliwe, ale przy odpowiednio silnych hasłach i właściwych algorytmach staje się praktycznie niewykonalne w rozsądnym czasie. patrz: shadow(5) - opis pliku haseł

Nadzorowanie dostępu w systemach uniksowych odbywa się za pomocą niepowtarzalnych identyfikatorów numerycznych przypisanych do każdego użytkownika (uid – ang. *user identifier*). Identyfikator grupy (gid – ang. *group identifier*) jest dodatkowym identyfikatorem, który można stosować do określania praw przynależnych więcej niż jednemu użytkownikowi. Kontrolowanie dostępu odnosi się do rozmaitych obiektów w systemie. Każdy plik osiągalny w systemie jest chroniony za pomocą standardowego mechanizmu kontrolowania dostępu. Ponadto inne obiekty używane wspólnie, takie jak sekcje pamięci dzielonej i semafory, korzystają z tego samego systemu dostępu.

Z każdym obiektem w systemie Unix, do którego dostęp jest kontrolowany przez użytkownika i grupę, jest skojarzony jeden identyfikator uid i jeden identyfikator gid. Procesy użytkowników też mają pojedyncze identyfikatory uid, ale mogą mieć więcej identyfikatorów gid. Jeżeli identyfikator użytkownika (uid) procesu zgadza się z identyfikatorem użytkownika obiektu, to proces ma prawa użytkownika, czyli właściciela (ang. *owner*) tego obiektu. W przeciwnym razie, jeżeli któryś grupowy identyfikator (gid) procesu pasuje do grupowego identyfikatora obiektu, to nadaje się prawa grupowe. W pozostałym przypadku proces ma do obiektu prawa „reszty świata", czyli wszystkich użytkowników nie będących właścicielem obiektu i nienależących do grupy obiektu.

System nadzoruje dostępy przez przypisywanie obiektom masek ochronnych, określających, które tryby dostępu (pisanie, czytanie lub wykonywanie) są udzielane procesom właściciela, grupy lub świata. Właściciel obiektu może mieć pełny dostęp do pliku (tj. prawo czytania, pisania i wykonywania), inni użytkownicy zrzeszeni w pewnej grupie mogą mieć prawo tylko do czytania natomiast pozostali użytkownicy mogą nie mieć w ogóle dostępu do pliku. Jedynym wyjątkiem jest uprzywilejowany identyfikator „super-użytkownika" (root) – reprezentującego administratora systemu. W klasycznym modelu DAC (Discretionary Access Control) procesowi z identyfikatorem tego użytkownika zwykle udziela się dostępu do dowolnego obiektu z obejściem zwykłej procedury sprawdzania dostępu. W praktyce współczesne systemy mogą dodatkowo stosować mechanizmy rozszerzające ten model, takie jak ACL czy obowiązkowe polityki kontroli dostępu (np. SELinux/AppArmor).

Prawa dostępu związane są z każdym obiektem systemu za pomocą atrybutów. Atrybuty dostępu, uid oraz gid, widoczne są na przykład w momencie wylistowania zawartości bieżącego katalogu za pomocą polecenia:

```
ls -l

-rw-rw-r-- 1 czesio prac 747592 Mar 12 20:01 plik1.txt
-rwxr-x--- 1 czesio prac 239837 Mar 12 20:01 plik2.txt
dr-xr-xr-x 3 czesio prac 4096 Mar 12 20:01 katalog1

0123456789 x użytkownik grupa rozmiar data_modyfikacji nazwa
```

Na pozycji 0 umieszczony jest znacznik określający typ obiektu. Znak „-" oznacza zwykły plik, „d" oznacza katalog, „l" link symboliczny. Następne 9 pozycji służy do zapisu masek dostępu kolejno dla użytkownika, grupy oraz reszty świata, po 3 znaki każda. Pierwszy znak „r", jeżeli występuje, oznacza prawo do odczytu obiektu, kolejny „w" - prawo do zapisu a ostatni „x" - prawo do wykonania. Jeżeli zamiast litery występuje znak „-", oznacza brak danego prawa dla użytkownika grupy lub reszty świata. W następnej kolejności znajdują się: liczba obiektów powiązanych z danym obiektem (1 w przypadku plików, 0 lub więcej w przypadku katalogów), właściciel obiektu, grupa, rozmiar, data oraz nazwa.

X-Windows

X-Windows jest graficznym interfejsem użytkownika stworzonym w latach 80 i wykorzystywanym w systemach z rodziny Unix. Precyzyjniej nazywa się go X Window System (X11).

System X tworzy okna, na których program może tworzyć obraz, oraz zajmuje się obsługą urządzeń wejściowych (myszki, klawiatury, tabletu). Serwer X może rysować tylko najprostsze obiekty (odcinki, wielokąty, elipsy, wyświetlać bitmapy, stawiać pojedyncze piksele), nie dostarcza natomiast żadnego interfejsu użytkownika, czyli przycisków, rozwijanych menu, pasków przewijania itp. Rysowaniem i obsługą tych elementów musi zająć się program, najczęściej jest to biblioteka widgetów. System X nie zajmuje się również obsługą okien, nie dostarcza żadnych wbudowanych mechanizmów do ich przesuwania, zmiany rozmiaru, zamykania i uruchamiania programów itd., nie rysuje także pasków tytułowych dla okien – tym wszystkim musi zająć się osobny program, tzw. menedżer okien (ang. *window manager*).

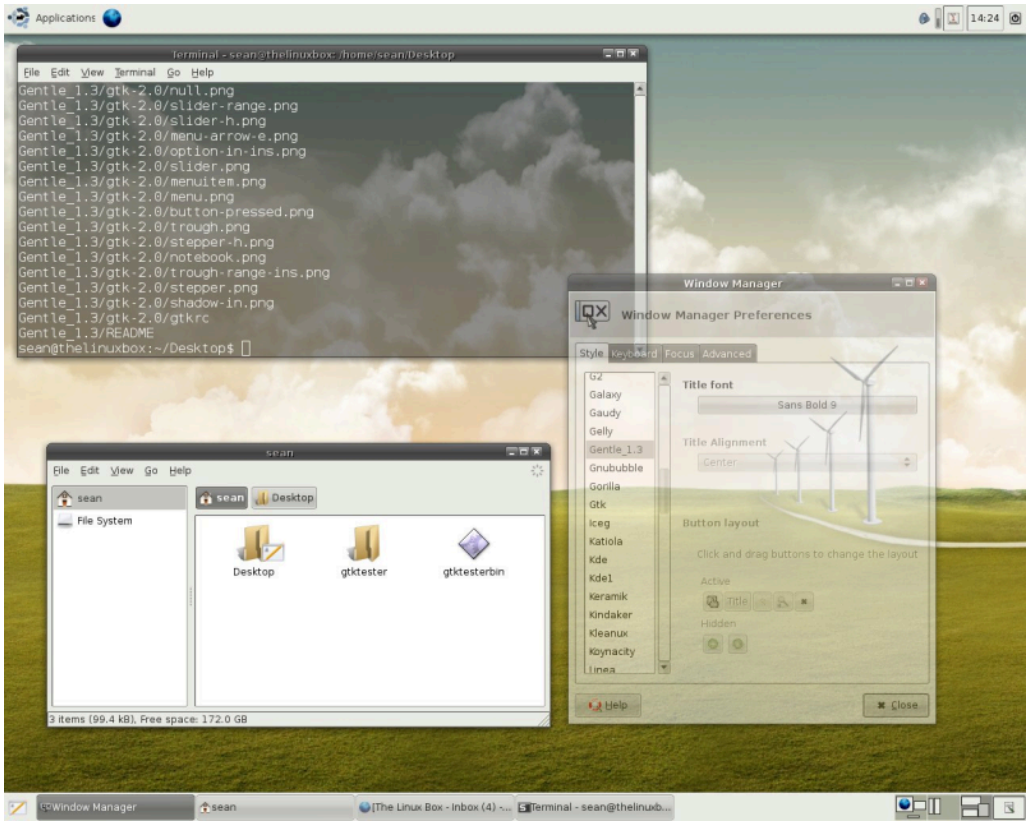
Pierwszymi bibliotekami widgetów były Athena, Motif oraz OpenLook, dzięki którym udało się stworzyć proste środowiska graficzne. Dzisiejsze GNOME i KDE (korzystające kolejno z: bibliotek GDK i GTK+ oraz Qt) to całe rozbudowane środowiska graficzne starające się tworzyć pewną użytkową całość.

X-Windows System zaprojektowany jest w architekturze klient-serwer, co oznacza, że składa się z dwóch elementów: z serwera (zajmującego się wyświetlaniem grafiki oraz obsługą klawiatury i myszy) oraz z klienta (czyli oprogramowania żądającego od serwera zmian zawartości okna). Oba elementy komunikują się ze sobą przy pomocy protokołu przezroczystego sieciowo, co oznacza, że nie ma znaczenia, czy oba elementy systemu działają na tym samym komputerze, czy na dwóch różnych maszynach. Jedna maszyna może wykonywać program, druga może wyświetlać wynik działania programu. Przykładem może być sytuacja, w której użytkownik pracuje zdalnie na odległym komputerze korzystając z X-Windows. Jego lokalny komputer (ten, przy którym siedzi) działa jako serwer X-Windows zapewniając graficzny interfejs użytkownika. Na zdalnym komputerze działa uruchomiony przez użytkownika program, będący klientem serwera X-Windows działającego na maszynie lokalnej. Z serwera na maszynie lokalnej program na maszynie odległej otrzymuje instrukcje sterujące użytkownika i na maszynę lokalną przesyła wyniki swojego działania.

Warto jednak zauważyć, że we współczesnych dystrybucjach Linuksa obserwujemy stopniową zmianę: obok X11 coraz częściej domyślną warstwą graficzną staje się Wayland. Ma to znaczenie praktyczne, bo część problemów historycznie „rozwiązywanych” w menedżerach okien i bibliotekach (np. bezpieczeństwo globalnych przechwyceń wejścia, integracja z nowoczesnymi GPU i kompozycją) jest w Wayland projektowana inaczej. W świecie Ubuntu przykładowo zmiany w domyślnych ustawieniach Waylanda były wprowadzane etapami (również z uwzględnieniem wsparcia sterowników). patrz: X11 support for Ubuntu 25.10

Nie oznacza to jednak, że X11 „zniknął z dnia na dzień”. Po pierwsze, część środowisk nadal domyślnie używa sesji X11 (np. XFCE w Xubuntu), a po drugie, nawet w sesji Wayland bardzo często uruchamiamy starsze aplikacje X11 przez warstwę kompatybilności XWayland. Dzięki temu użytkownik zwykle może przechodzić na Waylanda stopniowo, bez natychmiastowego porzucania starszego oprogramowania. patrz: Xubuntu 24.04 i Wayland

Jeżeli spojrzycie na rozwój dużych środowisk graficznych, zobaczycie, że trend jest wyraźny: Fedora od lat traktuje Waylanda jako domyślny wariant w GNOME, a w świecie KDE ogłaszano kolejne kroki w stronę pełnej migracji (wraz z planami wygaszania sesji X11 w przyszłych wydaniach). To jest dobry przykład, jak „system operacyjny” w praktyce ewoluuje: nie tylko w jądrze, ale również w kluczowych warstwach użytkowych, takich jak stos graficzny. patrz: Wayland jako domyślny w Fedora, patrz: KDE: Going all-in on a Wayland future



Przykładowy ekran X-Windows z uruchomionym środowiskiem XFCE.

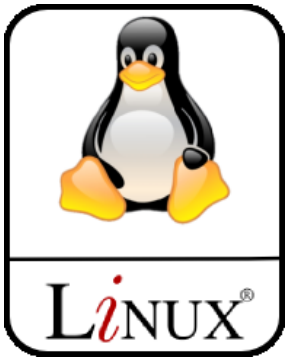
Podsumowanie

System Unix świetnie nadaje się jako system operacyjny dla różnej wielkości serwerów. Umożliwia on zarówno udostępnienie użytkownikom wielu usług sieciowych (takich, jak dostęp do sieci Internet, współużytkowanie plików, obsługa poczty elektronicznej), jak również zapewnia odpowiednie bezpieczeństwo. Stosując mechanizmy bezpieczeństwa zaimplementowane w Uniksie można zapewnić bezpieczeństwo usług świadczonych w sieci komputerowej, kontrolę dostępu do tych usług, prywatność zasobów gromadzonych przez użytkowników na serwerze oraz inne elementy wchodzące w skład tak zwanej polityki bezpieczeństwa. Cechy te sprawiają, że system Unix – w różnych swoich odmianach – znalazł zastosowanie w bankach, instytucjach finansowych, agendach rządowych i wielu innych obszarach wymagających niezawodności i bezpieczeństwa.

Przejrzysty projekt systemu Unix zaowocował wieloma naśladownictwami i modyfikacjami, z których w ostatnim czasie wyjątkowo aktywna stała się gałąź Linux.

1.2. Linux

Początek



Historia Linux-a sięga swoimi korzeniami systemu Unix. Wcześniej pisaliśmy o tym, że istnieje wiele różnych odmian Unixa. Jednym z jego klonów był także Minix - mały system, który choć nie miał wszystkich zalet Unix-a, jednak mógł być z powodzeniem używany na mniejszych maszynach klasy PC. Jego możliwości nie były wielkie - w zasadzie był wykorzystywany jedynie do uczenia studentów budowy systemów operacyjnych na żywym przykładzie z dostępnym kodem źródłowym. W zajęciach z tym systemem uczestniczył niejaki Linus Torvalds - fiński student z uniwersytetu w Helsinkach. Po głębokim (niewątpliwie) namyśle postanowił, że zamiast nieprzydatnego w praktyce, oraz ograniczonego licencyjnie Minix-a możnaby było stworzyć pełnoprawny system operacyjny zgodny z Unixami, ale dystrybuowany w zgodzie z zasadami wolnego oprogramowania. Rozpoczął prace nad takim systemem. Pierwsza wersja Linux-a (0.01) mogła obsługiwać dyski oraz mały system plików. Nie miała jednak jeszcze nawet zaimplementowanej obsługi napędu dyskietek. Krótko po pojawieniu się wersji 0.02, 5 października 1991 roku, Torvalds ogłosił istnienie systemu wraz z zaproszeniem do współpracy wszystkich chętnych. Wersja 0.02 posiadała już możliwość uruchomienia bash'a (Bourne Again Shell - popularna powłoka tekstowa) a także gcc (darmowa, genialna kolekcja kompilatorów, głównie wykorzystywana jako kompilator języka C i C++). Niestety poza tym prawie nic nie działało.

Lecz na grupie comp.os.minix pojawiły się pierwsze listy dotyczące Linux-a. Przynęta zadziałała - nowe jądro ściągnęło 10 osób, z czego pięć dokonało własnych poprawek. Linux zaczął się błyskawicznie rozwijać. Zaraz po tym jak wyszła wersja 0.03, pojawiła się 0.10, a w marcu 1992 roku - 0.95.

Pod koniec roku 1991 z systemu korzystało 100 osób, a samo źródło liczyło 10 tys. linii kodu! W I kwartale 1992 roku liczba użytkowników powiększyła się dziesięciokrotnie, a objętość kodu - ponad trzykrotnie. W grudniu roku 1993 pojawiła się wersja 0.99pl14 licząca 100 tys. linii i 20 tys. użytkowników, rok później pojawiło się pierwsze stabilne jądro oznaczone 1.0 oraz pierwsze firmy zajmujące się dystrybucją systemu - Red-Hat i Caldera.

Sama nazwa Linux nie jest dziełem Linusa choć pochodzi od jego imienia. Torvalds proponował nazwę FREAX. Jak twierdził zawierała w sobie słowo free (wolny), freak (dziwak) i X jak Unix. Lecz nazwa nie podobała się adminowi pierwszego serwera ftp na którym Linux był umieszczony - więc zaproponował nazwę Linux, zaaprobowaną przez Torvaldsa. Spontaniczny charakter prac oraz pełen dostęp do kodu źródłowego systemu wpłynęło na powstanie kilku wydań - ułatwiających instalację systemu, oraz łączących jądro z innymi użytecznymi programami. Z czasem wyłoniły się trzy główne historyczne odmiany (dystrybucje) systemu Linux: Debian, Red Hat oraz Slackware.

W 1999 roku nastąpiło wielkie zauroczenie systemem i gazety o tematyce komputerowej zaczęły coraz mocniej rozpisywać się na temat Linux-a. Wiele komercyjnych firm odkryło przy tym nowe, doskonałe źródło dochodu, a entuzjastyczne wypowiedzi na temat systemu Linux stały się normą. Od tego roku system stał się znany, rozpoznawalny oraz stał się poważną alternatywą dla rozwiązań komercyjnych. Dynamiczne tempo rozwoju systemu sprawia, iż staje się on z dnia na dzień coraz doskonalszy i bardziej powszechny. Ogromna rzesza programistów z całego świata, pracujących dla przyjemności, chętnie dokłada swoje trzy wartościowe grosze do tego, by system był coraz lepszy i wydajniejszy.

Należy przy tym pamiętać, iż nazwa Linux odnosi się raczej do samego jądra systemu, które jest dostępne na zasadach GNU General Public License (lub "copyleft") utworzonej przez Free Software Foundation. Celem tej organizacji, założonej przez Richarda Stallmana, jest promowanie "wolnego" oprogramowania o wysokiej jakości. Licencja GPL oznacza możliwość bezpłatnego kopiowania, użytkowania i rozpowszechniania binariów, jednak pod warunkiem, iż rozprowadzane są wraz z kodem źródłowym (dotyczy to także wersji komercyjnych Linux-a).

Dystrybucje Linux-a są zbiorem, w którym znajduje się zarówno jądro systemu, jak i potrzebne do pracy oprogramowanie. Różnią się one sposobem przygotowania systemu, mechanizmami łączącymi pakiety w jedną całość, programem instalacyjnym, przeznaczeniem oraz rodzajem i ilością dołączanych programów.

Początkowe idee, w świetle materialistycznej rzeczywistości współczesnego świata, trochę się niestety zacierają i nieuniknione, czysto komercyjne działania wielu firm tworzą nową erę rozwoju Linux-a... jaka ona będzie - zobaczymy. Należy pamiętać, że ducha darmowego oprogramowania tworzonego przez zapaleńców nie jest łatwo zniszczyć - świadczą o tym ciągle powstające w pełni niekomercyjne dystrybucje.

Linux nie jest jedynym przedstawicielem darmowych, otwartych systemów operacyjnych. Istnienie takich rozwiązań jak FreeBSD, BeOS czy GNU Hurd - daje nam wybór ... tak istotny dla zapaleńców (przecież muszą mieć jakiś temat do kłótni odnośnie wyższości świąt Bożego Narodzenia nad Wielkanocą ;)). Niemniej, ze wszystkich darmowych rozwiązań Linux jest najdojrzalszy i najbardziej "do przyjęcia" dla mniej doświadczonych użytkowników.

Dzisiaj

Dziś Linux ma już za sobą wiek młodzieńczych wad i stał się pełnoprawnym systemem operacyjnym. Można go wykorzystywać z powodzeniem jako zamiennik Windows czy MacOS w codziennej pracy czy w zastosowaniach domowych. Wyposażony w wygodny, okienkowy interfejs użytkownika, graficzne instalatory, olbrzymią ilość dokumentacji w internecie oraz sporą liczbę publikacji książkowych

pozwala na w miarę komfortową przesiadkę z innego systemu operacyjnego.



Podstawowy system Linux jest standardowym środowiskiem dla aplikacji i programów użytkownika, które jednak nie narzuca żadnych standardowych sposobów korzystania z dostępnych funkcji, pojmowanych jako całość. W skład tzw. dystrybucji wchodzi aktualnie znacznie więcej, niż to zwyczajowo wymaga się od systemu operacyjnego. W zasadzie każda dystrybucja jest kompletnym zestawem całego potrzebnego oprogramowania, poczynając od jądra, poprzez programy narzędziowe i użytkowe, oprogramowanie biurowe i graficzne, na narzędziach developerskich i serwerach wszystkich usług sieciowych kończąc. Aby zarządzać tym gigantycznym zbiorem programów powstała potrzeba zbudowania jeszcze jednej warstwy usług - tzw. systemu zarządzania pakietami. Dystrybucja Linux-a obejmuje wszystkie standardowe części systemu Linux wraz ze zbiorem narzędzi administracyjnych upraszczających wstępne instalowanie systemu oraz dalsze jego ulepszanie i konfigurowanie, jak również umożliwiających instalowanie i usuwanie dodatkowego oprogramowania. Większość z nich zawiera także własne narzędzia do administrowania systemem plików, tworzenia kont użytkowników i zarządzania nimi, administrowanie siecią itp...

W zastosowaniach domowych jego główną zaletą pozostaje cena - jest za darmo, bądź też w przypadku niektórych dystrybucji - za niewielką opłatą. Większość oprogramowania dla Linux-a jest również darmowa. Warstwa kompatybilności Wine (patrz WineHQ) pozwala uruchamiać część aplikacji Windows bez potrzeby instalowania drugiego systemu, a jej komercyjnym rozwinięciem jest CrossOver firmy CodeWeavers (patrz CodeWeavers CrossOver). Zazwyczaj jednak nie ma takiej potrzeby: przeciętny użytkownik znajdzie bez problemu darmowe odpowiedniki potrzebnych mu programów, w dodatku często zainstaluje je prościej dzięki repozytoriom pakietów.

Linux jest systemem, w którym bezpieczeństwo stoi wysoko, ale ja uważam, że warto tu mówić precyzyjnie: "bezpieczniejszy" nie znaczy "odporny na wszystko". W dystrybucjach standardem są mechanizmy takie jak firewall, aktualizacje bezpieczeństwa i szyfrowanie dysków (np. LUKS). Współcześnie linux wykorzystuje nowoczesne systemy plików, np. ext4, XFS czy Btrfs. A malware? Ono istnieje również na Linuksa (zwłaszcza w środowiskach serwerowych, w kontenerach i przy atakach na łańcuch dostaw), więc podstawą jest rozsądna konfiguracja systemu, minimalizacja usług, aktualizacje i kopie zapasowe.

Szeroko znana stabilność systemu odnosi się głównie do zastosowań serwerowych. Tam nie występuje problem zawieszania się systemu, niedostępności pewnych usług czy też konieczności przestojów serwisowych. Do normy należy nieprzerwana praca serwera liczona w latach. W zastosowaniach biurowo-domowych już nie jest aż tak różowo. Zarówno poszczególnym aplikacjom, jak i samym środowiskom graficznym, zdarza się zawiesić - nie częściej niż w przypadku Windows, ale się zdarza. Ponadto funkcjonalność darmowych aplikacji zazwyczaj jest mniejsza, są trudniejsze w obsłudze i nie wyposażone w rozbudowany system pomocy.

Jego olbrzymią zaletą jest fakt, iż każdy z Was, w każdej chwili może zupełnie za darmo sprawdzić, co potrafi ten system. Jest mnóstwo argumentów uzasadniających, dlaczego warto się z nim zapoznać. Linux ciągle szybko się rozwija i coraz więcej firm z niego korzysta. W przyszłości będzie więc zapewne łatwiej pozyskać pracę znając więcej, niż jeden system operacyjny. Warto także chociażby po to, aby kształtować i doskonalić swoją wiedzę informatyczną. W Linux-ie zrozumiesz na jakiej podstawie funkcjonuje sprzęt i jak ogólnie działa każdy (nie tylko Linux) system operacyjny.

Nie do pominięcia jest również satysfakcja związana z użytkowaniem systemu bardziej zaawansowanego i trudniejszego w obsłudze niż standardowy Windows. Jednak nie należy zapominać, że bywa on mniej dopracowany i "wygładzony" niż jego komercyjni konkurenci - Windows i MacOS. Wymaga wiedzy, wymaga od użytkownika świadomości własnych czynów, oraz czasem potrafi sprawić problemy z obsługą nowego sprzętu. Mimo pojawienia się graficznych konfiguratorów, ciągle czasami szczegółowe dostrojenie różnych usług i serwerów wymaga ręcznej edycji plików konfiguracyjnych. Tak więc - nie posiadając odpowiedniej wiedzy informatycznej, zapału i czasu można szybko zniechęcić się i już nigdy nie sięgnąć po ten system. Ale moim zdaniem warto spróbować - choćby po to by wiedzieć, że Wam nie odpowiada.

Jądro Linux

Jądro Linux (ang. *Linux kernel*) to najważniejsza i centralna część systemu operacyjnego Linux. Kod jądra Linux-a jest objęty licencją GNU General Public License - każdy może go pobrać z sieci, przejrzeć i modyfikować do woli (pomijając zastrzeżenie, że trzeba być naprawdę dobrym programistą by je zrozumieć).

Jądro Linux-a jest w dużym stopniu zgodne ze standardami ANSI i POSIX, obsługuje wielozadaniowość, wielowątkowość, wielobieżność (wielodostęp), pamięć wirtualną, biblioteki współdzielone, ładowanie na żądanie, współdzielony kod wykonywalny (ang. *copy-on-write*), dobre zarządzanie pamięcią i obsługę sieci TCP/IP. Współcześnie jest ono jądrem hybrydowym, choć wywodzi się z jądra monolitycznego (a więc w większej części ładowanym na stałe do pamięci przy uruchamianiu systemu) z ładowalnymi modułami. Sterowniki urządzeń i rozszerzenia jądra zwykle pracują w trybie uprzywilejowanym, z pełnym dostępem do sprzętu, nieliczne jednak działają w trybie użytkownika. W przeciwieństwie do typowych jąder monolitycznych, sterowniki urządzeń są zwykle kompilowane jako moduły, które można załadować i wyładować na działającym systemie. Podobnie, sterowniki mogą być wywłaszczone w określonych warunkach. Ta funkcja została dodana w celu poprawnej obsługi przerwań sprzętowych i systemów wieloprocesorowych.

Warto zwrócić uwagę, że sposób wydawania jądra zmieniał się na przestrzeni lat. Dziś nie obowiązuje już prosta zasada "parzyste stabilne, nieparzyste niestabilne". Zamiast tego mamy główną linię rozwoju (mainline), wydania stabilne oraz osobno gałęzie długoterminowo utrzymywane (LTS). Aktualne numery wersji można zawsze sprawdzić na stronach projektu (patrz kernel.org oraz opis modelu wydań: Active kernel releases).

Zazwyczaj dystrybucja dostarcza nam prekompilowane i skonfigurowane jądro do natychmiastowego zastosowania, niemniej - zawsze można je zamienić na takie, które skompilujemy i skonfigurujemy samodzielnie. Jest to jedno z pierwszych zadań, których podejmują się ludzie zaczynający swoją przygodę z Linux-em, a przy okazji - jedna z pierwszych porażek ;) Niemniej po lekturze paru artykułów w sieci, oraz serii czasochłonnych prób, każdy z Was, drodzy Studenci, jest w stanie przygotować sobie własny kernel, dostosowany do sprzętu na którym jest uruchomiony, jak garnitur szyty na miarę.

Graficzne środowiska użytkownika

W początkowych latach istnienia systemu główną powłoką (interfejsem użytkownika) była tekstowa konsola. Mimo jej wygody (zdecydowanie większej niż konsola Windows) dziś w zastosowaniach biurkowych większość osób wybiera środowiska graficzne. Równocześnie w administracji systemami i w pracy serwerowej powłoka tekstowa pozostaje narzędziem kluczowym. Dlatego też i w systemie Linux w zasadzie od samego początku pojawił się X-Windows. Sam X-Windows udostępnia jednak tylko podstawową obsługę okien, odbiegając funkcjonalnością od tego, do czego przyzwyczaił nas MacOS czy Windows. Dlatego w dniu dzisiejszym praktycznie nikt nie używa już X-Windows w czystej postaci, lecz w połączeniu z jednym z graficznych środowisk pracy. Warto też dodać, że we współczesnych dystrybucjach coraz częściej spotkacie sesje oparte o Wayland, a X11 bywa utrzymywany dla kompatybilności i specyficznych przypadków użycia (patrz Fedora: Wayland jako domyślna sesja GNOME).

Użytkownik przyzwyczajony do MS Windows może nie zdawać sobie sprawy, że Linux oferuje wybór również w tej dziedzinie, nie skazując nas na jedyny słuszny interfejs, przygotowany przez dostawcę systemu. Większość dystrybucji oferuje nam wprawdzie na początek jedno lub więcej wstępnie skonfigurowanych środowisk, lecz żadna nie odbiera nam prawa do instalowania innych, co pozwala na poszukiwanie pulpitu odpowiadającego nam najlepiej.

W Linux-ie sami możemy podjąć decyzję, czy chcemy pracować w środowisku o bardzo rozbudowanej funkcjonalności (jednak wymagającym najczęściej nowoczesnego sprzętu), czy może wystarczy nam prosty tzw. menedżer okien, oferujący tylko niezbędne funkcje, takie jak dostęp do menu aplikacji, zegar i zakładki z uruchomionymi programami. Nieważne czy lubimy bajery i błyski, czy raczej prosty i funkcjonalny desktop, na pewno znajdziemy coś dla siebie. Do tych prostych środowisk, paradoksalnie przeznaczonych raczej dla zaawansowanych użytkowników, możemy zaliczyć:

- **FVWM** - jeden z najstarszych menedżerów okien, uznawany za najbardziej konfigurowalny (oferuje nawet własny język skryptowy). Bardzo ładny temat/nakładkę na FVWM - FVWM-Crystal, rozwija Polak, Maciej Delmanowski;
- **Fluxbox**, **Windowmaker**, **IceWM** - trzy popularne, lekkie menedżery okien, oferujące jedynie najbardziej podstawowe funkcje, jednak dzięki temu pracujące wydajnie również na naprawdę starym sprzęcie;
- **XFCE** - stosunkowo młode środowisko, oferujące mniejszą funkcjonalność niż (opisane dalej) GNOME i KDE, jednak zdecydowanie mniej wymagające pod względem sprzętu. Fani XFCE cenią sobie w tym środowisku prostotę i elegancję.

Jednak zdecydowana większość użytkowników dokonuje wyboru pomiędzy dwoma głównymi konkurentami, GNOME i KDE. Oba stały się już osobnymi i potężnymi projektami programistycznymi, integrującymi międzynarodowych użytkowników. Oba projekty wychodzą z założenia dostarczenia użytkownikowi zintegrowanego systemu zarządzania i pracy w systemie operacyjnym. Oba posiadają pulpity, na których można umieszczać ikony, oba mają paski zadań, oba pozwalają na swobodne przełączanie się między aplikacjami oraz na dokonywanie podstawowych konfiguracji systemu. Oba także obsługują mechanizmy tematów (skórek) - dzięki którym można dostosować ich wygląd do własnych upodobań. Czym więc się różnią?

KDE



KDE jest najbardziej zaawansowanym technicznie środowiskiem użytkownika dla systemu Linux. W zasadzie stanowi system sam w sobie. Oprócz tego co znacie z Windows (a więc pulpitu, menu start, ikon, itp) KDE daje nam możliwość w zasadzie dowolnego konfigurowania ekranu, wyglądu i położenia poszczególnych elementów, skrótów klawiszowych, położenia przycisków, itp. Nie ma najmniejszego problemu by KDE wyglądało dokładnie tak samo jak Windows z jednej strony, z drugiej - upodobnienie go do MacOS także jest możliwe.

Przy tworzeniu KDE ogromny udział mieli ludzie o tzw. dobrym guście, co oznacza, że w większości przypadków KDE jest po prostu eleganckie. Ponadto istnieją także specjalne strony, takie jak www.kde-look.org, gdzie można znaleźć ogromną ilość dodatkowych tapet, ikon, motywów, efektów, wygaszaczy ekranu, i tak dalej.

Z technicznego punktu widzenia, KDE to nie tylko ładnie wyglądające okienka. To ogromna ilość rozwiązań, które ułatwiają jego używanie, takich jak wyświetlanie tekstu i grafiki na pulpicie, grupowanie okien jednej aplikacji, jednolity system pomocy i konfiguracji, pełna internacjonalizacja, dostęp do wielu pulpitów, i wiele innych. To także zestaw podstawowych aplikacji o świetnej jakości, jak np. zintegrowany system zarządzania informacjami osobistymi (ang. PIM, *Personal Information Manager*) Kontact - łączący obsługę poczty, kalendarza, listy zadań i książki adresowej, przeglądarka www i manager plików Konqueror, program do nagrywania płyt k3b, centrum obsługi muzyki Amarok, środowisko programistyczne KDevelop. Niektóre z aplikacji tworzonych dla KDE jeszcze nie osiągnęły poziomu innych, alternatywnych rozwiązań (np. KOffice jest mniej funkcjonalne niż LibreOffice), ale zawsze można znaleźć coś dla siebie. KDE udostępnia także system osadzania "aplikacji w aplikacji" KParts, co oznacza że programiści mogą łatwo w swoich programach wykorzystywać te już istniejące.

GNOME

GNOME to środowisko nieco skromniejsze w porównaniu z KDE, jeśli chodzi o funkcjonalność, jednak wiele osób preferuje je ze względu na bardziej przejrzysty interfejs i brak skłonności do przerostu formy nad treścią. Korzystając z niego, można stwierdzić, że w zasadzie ma te same funkcje, tylko wszystko w GNOME jest jakby skromniejsze, mniej krzykliwe, czasem może też mniej konfigurowalne, takie bardziej podobne do MacOS niż do Windows.

Liczba aplikacji pisanych specjalnie dla GNOME także jest mniejsza, co nie znaczy, że nie ma tych najważniejszych. W historii GNOME rolę PIM pełnił Evolution, a popularnymi przeglądarkami bywały różne projekty rozwijane na bazie silników webowych. Ponadto w GNOME można uruchomić aplikacje KDE (podobnie jak w KDE aplikacje pisane dla GNOME). GNOME jest także chętniej wybierane przez twórców poszczególnych dystrybucji jako podstawowe środowisko graficzne.

Dystrybucje Linux-a

Teoretycznie system Linux może zainstalować każdy, pobierając najnowsze wersje niezbędnych części systemu z ogólnie dostępnych serwerów ftp, po czym je kompilując. We wczesnych czasach systemu Linux czynność tę często musiał wykonać sam użytkownik. Jednak w miarę dojrzewania systemu poszczególne osoby i grupy dołożyły starań, aby zadanie to uczynić mniej bolesnym, dostarczając standardowego, wcześniej skompilowanego i skonfigurowanego, co więcej - spójnego - zbioru pakietów do łatwego instalowania.

Początkowo, dopiero co powstające dystrybucje można było porównać do zestawu archiwów. Każdy wchodzący w ich skład pakiet był po prostu archiwum, które odpowiedni program rozpakowywał i kopiował wchodzące w jego skład pliki do odpowiednich, na sztywno przypisanych katalogów. Współczesne dystrybucje mogą w tej dziedzinie znacznie więcej. Mogą dokonać konfiguracji pakietu, mogą automatycznie pobrać z internetu i zainstalować wymagane dla danego programu pakiety z bibliotekami (tzw. kontrola zależności), mogą samodzielnie skompilować kod źródłowy, dostarczają narzędzi wyszukiwania oprogramowania w repozytoriach sieciowych, i tak dalej ...

Pierwszą uznaną za kompletną dystrybucję był zbiór pakietów Linux-a nazwany SLS - ale to już daleka i głęboka historia. Poniżej znajdziecie krótki opis najistotniejszych dystrybucji współcześnie wykorzystywanych. Jest to jedynie wycinek całości - jednak wydaje nam się, że ujęliśmy w nim te najważniejsze.

Debian i pochodne

Debian

Sam **Debian** jest największym działającym do dziś projektem stworzenia wolnej dystrybucji systemu operacyjnego GNU/Linux przez ochotników na całym świecie, bez wsparcia jakiejkolwiek firmy (sic!) - wszystko opiera się na Umowie Społecznej Debiana oraz wolontariacie jego deweloperów. Debian cieszy się opinią stabilnego systemu o wysokiej jakości i łatwego do aktualizacji. Jest jednak często krytykowany za zbyt rzadkie wydania wersji stabilnych oraz za trudny w obsłudze instalator. Debian korzysta z pakietów DEB. Podstawowym instalatorem pakietów jest dpkg - instalator niskiego poziomu obsługiwany z linii poleceń, lub jego bardziej zaawansowany odpowiednik - APT, w którym wiele czynności jest zautomatyzowanych (pobieranie pakietów, rozwiązywanie zależności między pakietami). Do wygodniejszego zarządzania pakietami, Debian dysponuje nakładkami na powyższe narzędzia - dselect oraz nowszy - aptitude. Pierwsza wersja 1.x pojawiła się w 1996 roku.

Dla początkującego użytkownika Debian na pierwszy rzut oka może wydawać się trudny i odstraszający - ciągle ciąży na nim piętno systemu tworzonego "dla zapaleńców przez zapaleńców". Większość narzędzi systemu ciągle jest obsługiwana z konsoli i często przy jego rozwoju górę bierze podejście zachowawcze, ostrożne przy włączaniu do systemu nowoczesnych rozwiązań. Warto też zauważyć, że temat firmware (np. do kart Wi-Fi) został uporządkowany: Debian utworzył osobny komponent repozytorium *non-free-firmware* i może dołączać jego zawartość na nośnikach instalacyjnych w zależności od skonfigurowanych komponentów (patrz Debian Wiki: Firmware oraz komunikat instalatora: Debian Installer News). Z drugiej strony Debian jest postrzegany jako system stabilny, o przemyślanej strukturze pakietów i dużej dbałości o spełnienie współzależności między pakietami.

Ubuntu



Ubuntu (polska strona tutaj) i jego bliźniak - **Kubuntu** są (udaną) próbą "ucywilizowania" Debiana, dostosowania go do potrzeb początkującego użytkownika. Obie te dystrybucje są przeznaczone przede wszystkim do zastosowań biurowych i domowych (ang. *desktop*). Projekt sponsorowany jest przez firmę Marka Shuttlewortha Canonical Ltd. Samo słowo ubuntu pochodzi z języków plemion Zulu i Xhosa zamieszkujących południową Afrykę i oznacza "człowieczeństwo wobec innych" (ang. *humanity towards others*). Głównym celem dystrybucji jest dostarczenie zwykłym użytkownikom kompletnego, otwartego systemu, który będzie łatwy w obsłudze oraz jednocześnie stabilny, niezawodny i nowoczesny. Dodatkową zaletą jest fakt, że płyta instalacyjna jest zrobiona w stylu tzw. Live (co oznacza, że można z niej uruchomić system operacyjny i go "wypróbować" zanim się go zainstaluje). Ubuntu ma dwa główne typy wydań: LTS (co dwa lata) oraz wydania pośrednie (utrzymywane krócej). LTS otrzymuje 5 lat standardowego wsparcia bezpieczeństwa (z opcją wydłużenia), a wydania pośrednie są wspierane około 9 miesięcy (patrz Ubuntu release cycle). Jako współczesny przykład LTS można podać Ubuntu 24.04 "Noble Numbat", utrzymywane bezpieczeństwem do 31 maja 2029 (patrz Ubuntu 24.04 LTS release notes).

Ubuntu i jego pochodne korzystają z repozytorium pakietów Debiana, ale w przeciwieństwie do niego - wyposażono je w wygodne narzędzia instalacyjne oraz zarządzania systemem. Główna różnica między Ubuntu a Kubuntu to wykorzystywane środowisko graficzne: Ubuntu opiera się o GNOME, Kubuntu wykorzystuje KDE. Oprócz nich opracowano kilka innych wariantów, mniej popularnych, takich jak Xubuntu oraz Edubuntu.

Knoppix

Knoppix to kolejna dystrybucja Linux-a oparta na Debianie, możliwa do uruchomienia bezpośrednio z CD-ROM lub płyty DVD, bez instalacji na twardym dysku (LiveCD). Jest ona rozwijana przez niemieckiego inżyniera Klausa Knoppera. Podczas pracy Knoppix-a tworzony jest wirtualny system plików w pamięci RAM, natomiast wszystkie partycje na dysku twardym montowane są tylko do odczytu. Dlatego nie ma ryzyka przypadkowej utraty danych. Po uruchomieniu można go zainstalować na dysku za pomocą prostego w obsłudze instalatora Knoppix-installer.

Główną zaletą Knoppix-a jest wspomniana możliwość pracy po uruchomieniu z CD. Aktualnie jest wiele tego typu rozwiązań, ale to Knoppix był pierwszym szeroko znanym. Często wykorzystuje się go jako przenośne stanowisko pracy - wystarczy nam komputer z napędem CD oraz płyta z Knoppix-em i pamięć USB - by zawsze nasz system był z nami ;)

Inne dystrybucje

Red Hat

Red Hat – jedna z najważniejszych firm w świecie komercyjnego Linuksa. Historyczna dystrybucja Red Hat Linux była popularna m.in. dzięki systemowi pakietów RPM (ang. *Red Hat Package Manager*), który został przejęty przez wiele innych dystrybucji. Współcześnie głównym produktem biznesowym jest **Red Hat Enterprise Linux** (RHEL), a dystrybucją rozwijaną przez społeczność i sponsorowaną przez Red Hat jest **Fedora** (patrz Fedora and Red Hat Enterprise Linux).

Red Hat Enterprise Linux skierowany jest przede wszystkim do odbiorców komercyjnych takich jak przedsiębiorcy i firmy, dla których oferuje wsparcie i pomoc techniczną oraz aktualizacje w modelu subskrypcyjnym. W praktyce w wielu organizacjach ważniejsza od "nowinek" jest przewidywalność, długie wsparcie i kompatybilność - i właśnie to jest częścią oferty RHEL. Dla przykładu Red Hat deklaruje 10 lat wsparcia dla każdej głównej wersji w ramach subskrypcji (patrz RHEL subscription guide).

Fedora to dystrybucja rozwijana przez Projekt Fedora, sponsorowana przez Red Hat, z nowymi wydaniami pojawiającymi się zwykle co około pół roku. Jest często wybierana przez osoby, które chcą szybciej dostawać nowsze wersje środowisk graficznych i pakietów, jednocześnie pozostając w "ekosystemie" technologii, które później trafiają do rozwiązań enterprise (patrz Fedora and RHEL).

Arch Linux i pochodne

Jeżeli chcecie zobaczyć przykład dystrybucji, która konsekwentnie stawia na aktualność pakietów, minimalizm i podejście "zrób to sam", to warto przyjrzeć się **Arch Linux**. Jest to dystrybucja typu *rolling release*, której celem jest udostępnianie możliwie najnowszych stabilnych wersji oprogramowania, bez klasycznych "skoków wersji" całego systemu (patrz ArchWiki: Arch Linux oraz strona projektu: archlinux.org).

Arch jest intencjonalnie oszczędny na starcie: dostajemy podstawę systemu i to użytkownik decyduje, co dokładnie doinstalować i jak skonfigurować całość. Pakietami zarządza się narzędziem *pacman*, a dodatkowym źródłem oprogramowania jest społecznościowe repozytorium AUR. Dla jednych to wada (bo wymaga czytania i rozumienia), dla innych ogromna zaleta, bo system bywa dokładnie "taki jak chcę", bez zbędnych dodatków.

Wokół Archa powstało wiele dystrybucji pochodnych, które próbują zachować jego aktualność, a jednocześnie uprościć start (instalator, wstępna konfiguracja, gotowe profile). Jeżeli chcecie sprawdzić taki wariant, możecie rozejrzeć się po zestawieniu dystrybucji bazujących na Archu (patrz ArchWiki: Arch-based distributions). Ja lubię podawać Archa jako ciekawy kontrast do dystrybucji "spokojniejszych": pokazuje, jak wygląda Linux, gdy priorytetem jest świeżość i kontrola, a ceną bywa większa odpowiedzialność użytkownika.

Aurox

Aurox był tworzoną w Polsce dystrybucją systemu GNU/Linux. Przygotowali ją Robert Główczyński i Jarosław Górny. Wywodził się z dystrybucji Red Hat Linux (obecnie Fedora) i korzystał z pakietów RPM. Pierwsza wersja Auroxa ukazała się w listopadzie 2002 roku, regularne wydania zakończono w 2006 roku.

Gentoo



Gentoo - nazwa powstała od pingwina białobrewego, który po angielsku nazywa się *Gentoo Penguin*. Jest to dystrybucja raczej dla zaawansowanych użytkowników, nie polecamy jej jako pierwszego wyboru. Gentoo to w założeniu system najbardziej konfigurowalny i pozostawiający użytkownikowi pełną swobodę decyzji odnośnie każdej jego części. Dzięki systemowi zarządzania pakietami Portage cały system, lub tylko niektóre jego fragmenty mogą być automatycznie optymalizowane dla naszego i tylko naszego komputera - co zwiększa wydajność. O wyjątkowości Gentoo świadczy już sam proces instalacji. Polega on na niskopoziomowej konfiguracji wszystkich elementów systemu. Wybieramy więc podstawowe narzędzia systemowe np. program logujący, cron czy też jądro (w Gentoo mamy ich kilka do wyboru), a następnie instalujemy potrzebne oprogramowanie, głównie poprzez kompilację ze źródeł (choć niekiedy można też skorzystać z dostarczonych pakietów binarnych). Instalacja trwa długo - bywa że i kilka dni (sic!), jednak czas ten nie jest zmarnowany, gdyż Gentoo pozwala nam zbudować system dobrze dopasowany do naszych potrzeb. W zawiłościach niskopoziomowego procesu instalacji pomoże nam doskonały przewodnik użytkownika (również po polsku).

Inność i wyjątkowość Gentoo polega przede wszystkim na sposobie instalacji pakietów. Do tej pory omawiane dystrybucje korzystały z już skompilowanych programów (czyli takich, jakie otrzymujecie też w przypadku Windows - gotowe exe-ki w połączeniu z instalatorem). Gentoo natomiast korzysta z tego, że znakomita większość oprogramowania dla Linuxa to Open Source.

Sercem Gentoo jest Portage - potężne narzędzie, które w założeniu kompiluje każdy instalowany program na naszym komputerze bezpośrednio z kodu źródłowego, uwzględniając np. optymalizacje dokładnie dla naszego procesora. Jeśli porównamy to z klasycznym podejściem z innych dystrybucji - gdzie zazwyczaj otrzymujemy uniwersalne (czyli nieoptymalizowane) wersje programów - to można się domyśleć co będzie szybsze... Portage zapewnia również ciągłą aktualność, nie trzeba czekać na wydanie najnowszej wersji programu dla naszego systemu, ani na kolejne wersje systemu jako takiego. W przypadku Gentoo w zasadzie ciężko mówić o wersji - nie ma takowej, zawsze system jest po prostu najnowszy, o ile dba o to jego administrator.

Gentoo wymaga jednak posiadania pewnej wiedzy o GNU/Linux-ie, a także czytania (ze zrozumieniem) dokumentacji, przeglądania forum i "grzebania" w plikach konfiguracyjnych (w systemie nie ma tekstowych ani graficznych konfiguratorów). Nie obędzie się również bez poświęcenia zasobów systemu (głównie procesora i pamięci RAM) na częstą kompilację oprogramowania. Przy obecnych komputerach aktualizacja systemu nie przeszkadza jednak w korzystaniu z niego w środowisku graficznym. Gentoo nie jest dystrybucją dla każdego. Wymaga poświęcenia dużej ilości czasu i bycia świadomym tego co robimy i co chcemy uzyskać. Po pewnym czasie odwdzięcza się nam jednak wygodą zarządzania oprogramowaniem i prostą aktualizacją. Dobrze postawione Gentoo działa długo, jest to więc dobry wybór dla cierpliwych, świadomych i (wbrew pozorom) leniwych użytkowników komputerów 😉

Slackware



Slackware jest najstarszą aktywnie rozwijaną dystrybucją Linux-a, choć warto zauważyć, że Debian jest młodszy zaledwie o 31 dni. W 1992 Peter MacDonald założył firmę Softlanding Linux System, która wydała wspomnianą we wstępie SLS - pierwszą wszechstronną dystrybucję zawierającą takie elementy jak środowisko graficzne X czy protokół TCP/IP. Założony przez Patricka Volkerdinga Slackware oparty został w większej mierze właśnie o SLS. Pierwsza wersja, 1.00, została wydana 17 lipca 1993 roku. Slackware prezentuje odmienne podejście niż inne popularne dystrybucje, takie jak Debian, Ubuntu, Red Hat, Gentoo, SuSE. Najtrafniej można ją określić mianem "uniksowej", ze względu na politykę dostarczania jedynie stabilnych wersji oprogramowania, organizację skryptów startowych w stylu BSD (ang. *Berkley Software Distribution* - jedna z wersji "czystego" Unixa) i brak dedykowanych narzędzi konfiguracyjnych wyposażonych w GUI, które można znaleźć w innych odmianach Linux-a. Realizacja zarządzania pakietami w Slackware również jest nietypowa ale zgodna z zasadami minimalizmu i prostoty. Pakiety są to zwykłe archiwa tar spakowane programem gzip, zawierające dodatkowe informacje o ich zawartości oraz opcjonalnie skrypt instalacyjny. System ten obsługuje instalację, uaktualnianie i usuwanie pakietów równie prosto, jak w innych dystrybucjach. Nie usiłuje jednak śledzić czy pilnować tego, co określa się mianem zależności (tj. upewnienia się, że system posiada wszystkie biblioteki i programy, których nowy pakiet potrzebuje do prawidłowego działania).

SuSE



SUSE, prosta w obsłudze dystrybucja Linux-a jest przeznaczona zarówno dla niedoświadczonego użytkownika jak i doświadczonych znawców tego systemu. Oryginalnie SUSE Linux była niemieckim tłumaczeniem dystrybucji Slackware Linux. S.u.S.E został założony w 1992 roku jako grupa konsultingowa, która spośród wielu innych rzeczy regularnie wydawała nowe wersje oprogramowania zarówno dla SLS jak i Slackware oraz pisała instrukcje obsługi dla systemów UNIX/Linux. Pierwszy raz CD zawierający SLS/Slackware został wydany dwa lata później w marcu pod dźwięczną nazwą S.u.S.E Linux 1.0 z jądrem linux 1.0. Z biegiem czasu wprowadzono do systemu kilka rozwiązań firmy Red Hat Linux, np. pakietowanie typowe dla Slackware (tar.gz) zastąpiono pakietami typu RPM.

SuSE oryginalnie była akronimem niemieckich słów *Gesellschaft für Software und Systementwicklung Gmbh* (ang. *Company for Software and System Development Ltd*). Od momentu kiedy została przejęta przez firmę Novell, oficjalna nazwa tego systemu brzmi “SUSE Linux”. Istnieje jeszcze nieoficjalna plotka, że nazwa została nadana na cześć niemieckiego pioniera komputerowego Konrada Zuse. Rozwijana w Niemczech dystrybucja w styczniu 2004 roku została przejęta przez Novella. Do wersji 7.0 w system włączona była obsługa języka polskiego; została przywrócona dopiero przez Novella. Dystrybucja zawiera graficzny program instalacyjny, narzędzie do konfiguracji systemu i zarządzania pakietami w jednym o nazwie YaST2. Doświadczeni użytkownicy znajdą wiele aplikacji umożliwiających programowanie w różnych językach, w tym Mono (odpowiednik środowiska .net firmy Microsoft). SUSE jest dostępne w dwóch wersjach - do pobrania za darmo oraz do zakupu w wersji box, zawierającej pakiety komercyjne oraz instrukcję obsługi i kilkadziesiąt dni pomocy technicznej przy instalacji. Świeżo zainstalowany system nie wymaga konfiguracji i jest gotowy od razu do pracy – co czyni tę dystrybucję przyjazną nowicjuszom. Na rynek wyszły bowiem różne wersje tego systemu, przeznaczone zarówno dla firm jak i dla komputerowych laików. Wersją dla firm jest SUSE Linux Enterprise Desktop lub jego wersja serwerowa SUSE Linux Enterprise Server. Do zastosowań domowych poleca się openSUSE, tak w wersjach pobieranych z Internetu jak i pudełkowych, zawierających na płytach instalacyjnych oprogramowanie o zamkniętym kodzie. Dystrybucja jest bardzo popularna i używana głównie jako system biurkowy. Warto również wspomnieć o bardzo dobrej obsłudze laptopów.

Podsumowując, SUSE jest dobrym narzędziem dla ludzi, którzy chcą po prostu zainstalować system i mieć święty spokój, których nie interesuje zbytnio konfiguracja, strojenie parametrów i częste uaktualnienia, i raczej nie grzebią po sieci w poszukiwaniu ciągle nowych programów.

PLD



Kolejną polską i tworzoną głównie przez Polaków dystrybucją Linuxa jest PLD. Jednakże jest ona zupełnie inna niż konkurencyjny Aurox. Po pierwsze - powstała znacznie wcześniej. Po drugie - nie jest klonem jakiejkolwiek wcześniejszej dystrybucji, tylko projektem opracowanym od początku. W PLD można znaleźć wiele połączonych rozwiązań pochodzących z

różnych dystrybucji, przykładowo do zarządzania pakietami można wykorzystać zarówno RPM (jak w Red Hat Linux, jednak wykorzystanie jego możliwości jest znacznie efektywniejsze), jak i APT (z Debiana) czy też własny system zarządzania pakietami i zależnościami poldek.

W przeciwieństwie do Auroxa, PLD nie jest przeznaczona dla początkujących. Jest trudna, przeznaczona głównie do zastosowań serwerowych, dodatkowo - bardzo bezpieczna. Ogromny nacisk twórcy położyli na zabezpieczenie systemu przed włamaniem oraz na obsługę najnowszych technologii sieciowych (przykładowo - PLD jako pierwsza używa protokołu IP w wersji 6 jako domyślnego). Konfiguracja większości usług i programów jest możliwa jedynie przez edycję plików konfiguracyjnych.

Co wybrać?

Czytając powyższy opis można dojść do wniosku, że ... *osiołkowi w żłoby dano*... Którą dystrybucję wybrać?

Dobrej odpowiedzi na to pytanie nie ma, a w zasadzie jest jedna, nic nie mówiąca - wybierz tę, która Ci najbardziej odpowiada ;)

Możemy jednak udzielić paru wskazówek:

- jeśli chcesz tylko zobaczyć Linux-a w działaniu i jesteś "oporny" na nowości - ja jako pierwszy wybór wskazałbym Ubuntu w trybie Live (możliwość "Wypróbuj" bez instalacji), bo jest nastawione na łatwy start i ma bardzo dużo poradników w sieci. W podobnym stylu działa też wiele innych dystrybucji oferujących tryb Live;
- jeśli chcesz spróbować korzystać z Linux-a na dłuższą metę, ale nie wiesz nic o systemach Unixowych - możesz zainstalować Ubuntu (najlepiej wydanie LTS), Fedorę albo openSUSE na twardym dysku. Każdy z nich da się zainstalować obok Windows i używać z nim na zmianę. Każdy jest wyposażony w sympatyczny, graficzny instalator, a proces instalacji nie jest trudniejszy niż instalacja Windows. Który z nich? Ubuntu jest często wybierane jako bezpieczny start, Fedora bywa wybierana przez tych, którzy chcą świeższe pakiety, a openSUSE przez tych, którzy cenią narzędzia administracyjne i porządek w konfiguracji;
- jeśli zależy Ci na możliwie aktualnych wersjach oprogramowania i lubisz mieć kontrolę nad tym, co dokładnie masz zainstalowane - zerknij na dystrybucje z rodziny Arch Linux (Arch i pochodne), ale pamiętaj, że rolling release wymaga większej odpowiedzialności i czytania komunikatów przy aktualizacjach (patrz ArchWiki: Arch Linux);
- jeśli jesteś zainteresowany dogłębnym poznaniem działania systemów operacyjnych ogólnie, a w szczególności Linux-a - polecamy Gentoo ... sam proces instalacji nauczy Cię wiele;
- jeśli szukasz stabilnego systemu na serwer i zależy Ci na komercyjnym wsparciu - dobrym wyjściem może okazać się wybór Red Hat Enterprise Linux w modelu subskrypcyjnym (patrz RHEL subscription guide). Debian również świetnie się sprawdzi - pod warunkiem odpowiedniej wiedzy administratora.

Powyższe uwagi są jedynie wskazówkami. Ostateczny wybór i tak będzie należał do Was - i to właśnie jest zgodne z duchem Linux-a 😊

Na koniec warto wspomnieć o **WSL** (*Windows Subsystem for Linux*) – mechanizmie, który pozwala uruchomić środowisko GNU/Linux bezpośrednio na Windows, bez klasycznego dual-bootu i bez ręcznego zarządzania maszyną wirtualną. Jest to rozwiązanie stworzone głównie z myślą o pracy deweloperskiej i administracyjnej: macie do dyspozycji narzędzia linuksowe, powłokę i wiele aplikacji uruchamianych „obok” świata Windows. W praktyce funkcjonują tu dwie generacje: WSL 1 (warstwa translacji wywołań systemowych) oraz WSL 2, który używa rzeczywistego jądra Linuksa uruchamianego w lekkiej maszynie wirtualnej zarządzanej przez system (patrz Microsoft: co to jest WSL; patrz Microsoft: porównanie WSL 1 i WSL 2).

Z punktu widzenia użytkownika WSL bywa świetnym kompromisem: możecie np. pracować w Windows, a jednocześnie korzystać z linuksowych narzędzi, kompilatorów i skryptów, instalując wybraną dystrybucję (np. Ubuntu, Debian, openSUSE, Kali czy Arch) i uruchamiając ją poleceniem systemowym. Microsoft promuje prostą instalację z poziomu konsoli (np. *wsl --install*) oraz model aktualizacji niezależny od całego systemu Windows (patrz Microsoft: instalacja WSL). Zauważcie jednak, że mimo wygody nie jest to „pełnowymiarowy serwer” ani zamiennik klasycznego Linuksa w każdym zastosowaniu: część scenariuszy sieciowych i wydajnościowych zależy od tego, czy używacie WSL 1 czy WSL 2, oraz jak trzymacie pliki (w systemie plików Linuksa czy na partycji Windows). WSL jest więc świetnym narzędziem do nauki i pracy developerskiej, ale w krytycznych wdrożeniach nadal częściej spotkacie natywny Linux albo klasyczne wirtualizacje/kontenery.

1.3. Windows

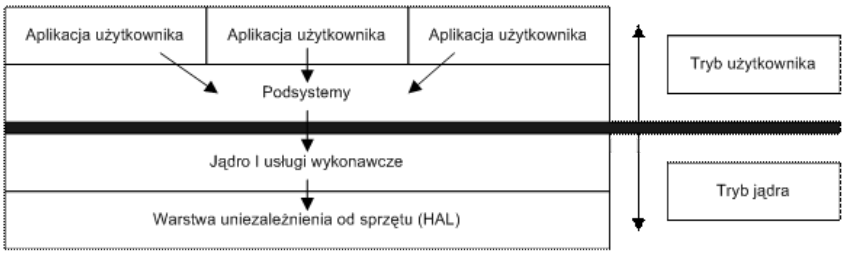
Windows 2000



W lutym 2000 roku, Microsoft wprowadził na rynek system Windows 2000, który rozwijany był pod nazwą Windows NT 5.0. Z powodzeniem trafił on zarówno do segmentu serwerowego, jak również na rynek stacji roboczych. Najbardziej znaczącą funkcjonalnością nowych Okienek była usługa Active Directory, która miała zastąpić model domeny Windows Server z systemu NT 4.0, a zbudowana została w oparciu o technologie DNS, LDAP oraz Kerberos.

Wiele funkcji zaczerpnięto z systemu Windows 98, jak na przykład ulepszonego menedżera urządzeń, odtwarzacz Windows Media Player czy DirectX, dzięki czemu po raz pierwszy wiele nowych gier można było uruchomić na jądrze NT. System doczekał się wsparcia dla DirectX 9.0c, co w tamtym okresie było istotne dla kompatybilności z grami (warto jednak pamiętać, że nowsze generacje DirectX (10 i wyżej) weszły dopiero wraz z Windows Vista i kolejnymi wersjami systemu; patrz Microsoft: DirectX End-User Runtimes (June 2010)).

Windows 2000 to 32-bitowy, nowoczesny, wielozadaniowy z wywłaszczaniem, wielowątkowy system operacyjny z serii NT. Jako że system był opracowany w wersjach dla różnych procesorów, dużą wagę przyłożono do odpowiedniego wydzielenia obsługi sprzętu - tak powstała warstwowa architektura systemu wraz z warstwą uniezależnienia od sprzętu (HAL – *Hardware Abstraction Layer*).



Podstawowa architektura Windows 2000

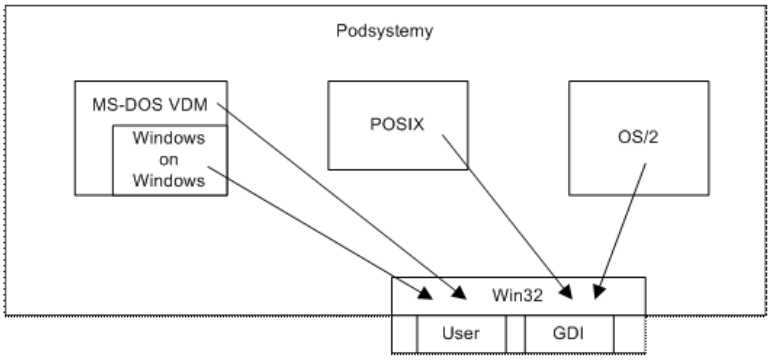
W trybie jądra uruchamiane są usługi wykonawcze, w trybie użytkownika - programy użytkownika. Usługi trybu jądra są chronione przez procesor, a usługi trybu użytkownika - przez system operacyjny. W momencie, gdy ochrona zawodzi, następuje wyjątek trybu jądra, co przedstawia się jako znany niebieski ekran. W Windows 2000 wiele usług i procesów systemowych działa w trybie użytkownika, ale kluczowe elementy (jądro, executive, sterowniki) pracują w trybie jądra; istotne jest też to, że od czasów Windows NT 4.0 duża część obsługi USER/GDI została przeniesiona do przestrzeni jądra (win32k.sys) z powodów wydajnościowych (patrz Microsoft (archiwum MSDN): wzmianki o GDI i kontekście kernel-mode). Programy trybu użytkownika komunikują się z jądrem poprzez specjalne interfejsy programowe aplikacji (API). Znaczna część systemu (jądro, executive, sterowniki) była tworzona głównie w języku C, z użyciem asemblera tam, gdzie wymagały tego fragmenty zależne od architektury; z kolei elementy interfejsu i grafiki w większym stopniu korzystały z C++.

Z punktu widzenia architektury jądro jest centralną częścią systemu operacyjnego. Jądro Windows 2000 wywodzi się z modelu mikrojądra, lecz w praktyce ze względu na wydajność, swoją wewnętrzną budową bardziej przypomina jądro hybrydowe.

System operacyjny musi spełnić wiele różnych zadań, aby zapewnić aplikacjom odpowiednie środowisko pracy. Model mikrojądra narzuca wykonywanie przez jądro tylko podstawowych działań, takich jak przełączanie zadań. Pozostałe usługi nadal są realizowane, ale już poza obszarem jądra. Jedną z tych wyniesionych poza jądro usług jest egzekutor systemu (Windows 2000 Executive) skupiający w sobie olbrzymi zbiór różnych funkcji przeznaczonych do obsługi odrębnych zadań. Egzekutor pracuje w całości w trybie jądra. Posiada pełny dostęp do jądra i urządzeń wejścia-wyjścia. Większość aplikacji użytkownika nie korzysta z egzekutora bezpośrednio, polegając na podsystemie.

Podsystem może być postrzegany jako środowisko stykające się z jądrem i aplikacjami. Umożliwia uruchamianie programów, zapewniając im potrzebne usługi i interfejsy programowe API, równocześnie odwołując się do jądra z prośbą o wymagane zasoby. Podsystem jest właściwie tłumaczem pomiędzy grupą aplikacji a jądrem. Występują dwa rodzaje podsystemów. Podsystem Win32 jest klasą sam dla siebie, wszystkie inne podsystemy są właściwie tłumaczami między emulowanymi przez nie podsystemami a Win32.

Win32 jest najczęściej używanym i najistotniejszym podsystemem. Pierwotnie był takim podsystemem jak każdy inny, lecz ze względu na to, że korzystają z niego wszystkie funkcje użytkownika, niektóre jego elementy pracują w trybie jądra, będąc częścią egzekutora. Są to biblioteki User i GDI, sterujące odpowiednio interfejsem użytkownika i monitorem graficznym.



Podsystemy w Windows 2000 Server

Podsystem Win32 zawiera również podsystemy MS-DOS i 16-bitowego Windows (starsze systemy Windows 3.1 i 3.11), co pozwala na uruchamianie w specjalnej maszynie wirtualnej DOS programów MS-DOS.

Podsystem POSIX planowo miał umożliwiać uruchamianie aplikacji systemu Unix pod kontrolą Windows. POSIX (*Portable Operating System Interface*) jest standardem określającym usługi systemowe i wywołania API innych systemów operacyjnych dostępnych aplikacjom.

Do rodziny systemu Windows 2000 wchodzi 6 różnych produktów:

Windows 2000 Terminal – środowisko udostępniane przez Windows 2000 Server jako usługi terminalowe ze współdzielonymi procesami i pamięcią.

Windows 2000 Professional – prosta wersja systemu wykorzystywana raczej jako stabilna wersja „domowa” (konkurencja dla Windows Millennium Edition). System potrafi obsłużyć maksymalnie 2 procesory i 4 GB RAM.

Windows 2000 Server – podstawowa wersja „serwerowni” z rodziny 2000. System obsługuje maksymalnie 4 procesory i 4 GB RAM, zawiera Active Directory, Intellimirror, Kerberos oraz usługi terminalowe.

Windows 2000 Advanced Server – wersja analogiczna do wersji Server co do zawartości, potrafi obsłużyć do 8 procesorów i 8 GB RAM.

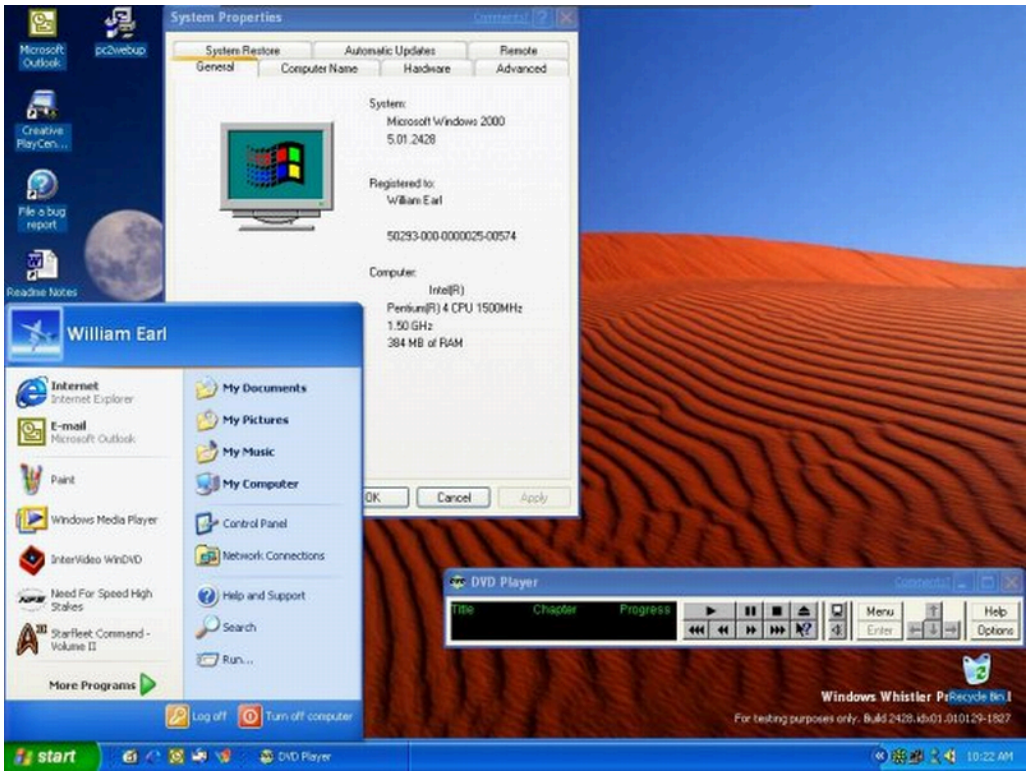
Windows 2000 Datacenter Server – analogicznie do wersji "Advanced Server", obsługuje maksymalnie 32 procesory i 64 GB RAM.

Windows 2000 Advanced Server, Limited Edition - specjalna wersja systemu wydana w 2001 r. przeznaczona dla procesorów Itanium firmy Intel.

Systemy rodziny Windows 2000 stosowane były często do użytku domowego w wersji Professional, jednak głównym celem ich zastosowania, a dokładniej wersji Server i wyższych, były małej i średniej wielkości serwery. System ten był dość prosty w instalacji i obsłudze. Zarządzanie serwerem odbywało się głównie przez graficzne środowisko.

Windows XP

Microsoft Windows XP (nazwa robocza Whistler) to wersja systemu operacyjnego Windows z rodziny Microsoft Windows NT (oparty na jądrze NT, następca Windows 2000) firmy Microsoft, wydana oficjalnie 25 października 2001 roku. Nazwa XP pochodzi od angielskiego słowa *eXPerience*, czyli doświadczenie, doznanie. Ta wersja systemu była przeznaczona głównie dla użytkowników indywidualnych potrzebujących wygodnego i wydajnego systemu operacyjnego dla swojego komputera domowego. Do systemu XP wprowadzono wiele nowych elementów i ulepszeń, jak choćby wbudowana zapora ogniowa chroniąca komputer przed atakami z sieci. Windows XP uważany był za system stabilny i wygodny w użytkowaniu, i jeszcze długo po oficjalnym zakończeniu wsparcia przez firmę Microsoft można go było spotkać na komputerach użytkowników.



Wygląd przykładowego ekranu w systemie MS Windows XP.

Następcą Windows XP była wersja Vista wypuszczona w styczniu 2007 roku. Nie zyskała ona uznania użytkowników; był to duży skok technologiczny (m.in. zmiany w modelu sterowników i mechanizmach bezpieczeństwa), a w połączeniu z ówczesnym sprzętem i kompatybilnością aplikacji przełożyło się to na słabszy odbiór. Wsparcie dla Windows Vista zakończyło się 11 kwietnia 2017 (patrz Microsoft Lifecycle: Windows Vista).

Windows 7

Wersja 7 systemu Windows pojawiła się na rynku w 2009 jako następca systemu Vista. System był dystrybuowany w 5 wersjach (jedna w dwóch wariantach):

- Windows 7 Starter – jest to najbardziej podstawowa wersja systemu Windows 7. Preinstalowana na komputerach o słabszych parametrach, głównie na netbookach. Dystrybucja i preinstalacja była prowadzona przez partnerów OEM – system był dostarczany wraz ze sprzętem. Wersja ta miała udoskonalony pasek zadań i mogła dołączyć do istniejącej grupy domowej (ang. HomeGroup).
- Windows 7 Home Basic – następna wersja systemu Windows 7, która także była dostarczana wraz ze sprzętem przez partnerów OEM. Jest to wersja przeznaczona na rynki rozwijające się. Zawiera dodatkowo Windows Mobility Center, które pozwala m.in. tworzyć połączenia sieciowe typu ad hoc oraz z urządzeniami przenośnymi. Pozwala na udostępnianie połączenia internetowego innym komputerom w sieci (ang. Internet Connection Sharing, ICS) oraz zawiera bogatszy interfejs graficzny.
- Windows 7 Home Premium – wersja przeznaczona dla użytkowników domowych, zarazem pierwsza, którą można kupić jako osobny produkt (do samodzielnej instalacji). Pozwala nie tylko dołączyć do grupy domowej, ale także tworzyć takie grupy. Zawierała część funkcji multimedialnych ze wszystkich dostępnych w różnych wersjach, m.in. Media Center i bardziej rozbudowaną obsługę plików multimedialnych. Inne funkcje to m.in. interfejs Aero Glass, obsługa technologii MultiTouch oraz rozpoznawanie pisma odręcznego.
- Windows 7 Professional – następna wersja możliwa do kupienia, kierowana do bardziej wymagających użytkowników, ale także małych firm. Wersja ta pozwala dołączyć do domeny, obsługuje szyfrowanie EFS, pliki offline, zaawansowane opcje kopii zapasowej – czyli jedne z ważniejszych funkcji przeznaczonych do pracy w sieci firmowej. Z mniej ważnych można wskazać możliwość podłączenia się przez Remote Desktop do innej stacji roboczej czy Location Aware Printing (szczególnie przydatna w komputerach przenośnych – automatycznie rozpoznaje, do jakiej sieci podłączony jest komputer, np. domowej czy firmowej, i odpowiednio ustawia domyślną drukarkę).
- Windows 7 Enterprise – wersja sprzedawana dużym przedsiębiorstwom (klienci „enterprise”). Do nabycia jedynie poprzez kanał Microsoft Volume Licensing. Obsługuje techniki: DirectAccess (działanie podobne do VPN), BitLocker i BitLocker To Go (pomaga chronić dane na komputerach i modułach pamięci przenośnej), BranchCache (skraca czas oczekiwania na pobranie pliku w sieci), AppLocker (określa możliwe do uruchomienia aplikacje). Wersja ta obsługuje tzw. MUI (ang. Multilingual User Interface), czyli inne wersje językowe systemu, i VDI (ang. Virtual Desktop Infrastructure).
- Windows 7 Ultimate – wersja o możliwościach wersji Enterprise, jednak możliwa do nabycia w kanałach konsumenckich (nie wymaga Volume Licensing).

Z technicznego punktu widzenia Windows 7 nie zmieniał zasadniczo technologii stosowanych już w XP i Vista - podstawową różnicą było to, że system był postrzegany jako stabilniejszy i szybszy. Miłe dla oka dodatki, takie jak interfejs Aero, czy ulepszone zarządzanie oknami, przycisk Start, nie zmieniają faktu, iż wersja 7 była raczej ewolucją niż rewolucją.

Windows 7 przez wiele lat był jedną z najpopularniejszych wersji tego systemu, ale z czasem przeszedł do historii: wsparcie dla Windows 7 zakończyło się 14 stycznia 2020 (patrz Microsoft Lifecycle: Windows 7). Dla porządku: w styczniu 2026 globalnie dominował Windows 11 (ok. 62%), a Windows 10 miał ok. 36% udziału wśród desktopowych wersji Windows (patrz StatCounter: udział wersji Windows).

Windows 8 i 8.1

Po sukcesie Windows 7 Microsoft spróbował mocniej odpowiedzieć na rosnącą popularność urządzeń mobilnych i dotykowych. W Windows 8 (2012) wyraźnie przesunięto akcent w stronę interfejsu projektowanego pod ekrany dotykowe: pojawił się nowy ekran startowy z kafelkami oraz środowisko aplikacji „nowego typu" (wówczas mocno kojarzone z Windows Store). Dla części użytkowników komputerów stacjonarnych była to zmiana trudna do zaakceptowania, bo ograniczała wygodę pracy myszą i klawiaturą. Zauważcie, że pod spodem system rozwijał się dalej dość konsekwentnie (wydajność, uruchamianie, sterowniki), ale to właśnie interfejs najbardziej wpłynął na odbiór tej wersji.

Windows 8.1 (2013) był próbą „dokręcenia śruby w dobrą stronę": część decyzji interfejsowych złagodzono, przywracając bardziej klasyczne mechanizmy nawigacji, poprawiając też spójność ustawień i konfiguracji. Niemniej w praktyce była to linia, która nie utrzymała długotrwałej dominacji na rynku. Wsparcie dla Windows 8.1 zakończyło się 10 stycznia 2023 (patrz Microsoft Lifecycle Announcement: Windows 8.1; patrz Microsoft Support: koniec wsparcia Windows 8.1).

Windows 10

Windows 10 trafił na rynek 29 lipca 2015 roku i był zapowiadany jako powrót do rozwiązań bardziej „komputerowych" po eksperymentach z Windows 8. Wprowadzono m.in. klasyczniejsze menu Start (w nowej postaci), nowe mechanizmy bezpieczeństwa (np. Windows Hello), nowe podejście do przeglądarki (Microsoft Edge), a także silny nacisk na spójność platformy pomiędzy różnymi typami urządzeń. Warto pamiętać o jeszcze jednym aspekcie: przez pewien czas system był oferowany jako bezpłatna aktualizacja dla użytkowników Windows 7 i 8.1, co miało ogromne znaczenie dla jego szybkiej adopcji (patrz Windows Experience Blog: dostępność Windows 10).

W przypadku Windows 10 bardzo istotna była zmiana modelu rozwoju: Microsoft konsekwentnie promował ideę „Windows as a Service", czyli systemu aktualizowanego cyklicznie nie tylko poprawkami bezpieczeństwa, ale też dużymi wydaniami funkcjonalnymi (wersje typu 1511, 1607, 1709, 21H2, 22H2 itd.). Dla administratorów i firm oznaczało to nowe podejście do planowania wdrożeń i cyklu życia wersji, a dla użytkowników domowych – znacznie częstsze aktualizacje, niekiedy odczuwalne jako wymuszane. Ten model był porządkowany kanałami obsługi/serwisowania i polityką wsparcia poszczególnych wydań (patrz Microsoft Learn: overview Windows as a service).

Z biegiem czasu Windows 10 stał się „systemem domyślnym" dla ogromnej liczby komputerów, szczególnie w firmach, gdzie liczy się kompatybilność oprogramowania i przewidywalność środowiska. Warto jednak jasno powiedzieć, że ta linia także ma swój koniec: Windows 10 osiągnął koniec wsparcia 14 października 2025, a wersja 22H2 była ostatnim wydaniem tej rodziny (patrz Microsoft Lifecycle: Windows 10 Home and Pro; patrz Microsoft Support: koniec wsparcia Windows 10).

Windows 11

Windows 11 stał się dostępny od 5 października 2021 roku i od początku był pozycjonowany jako system z mocnym naciskiem na „nowocześniejszą" warstwę użytkową oraz bezpieczeństwo bazujące na funkcjach sprzętowych. Z perspektywy użytkownika najbardziej widoczne były zmiany w interfejsie (przeprojektowane menu Start, inny układ paska zadań, szersze wykorzystanie mechanizmów organizacji okien), ale pod spodem istotne stały się również wymagania sprzętowe i mechanizmy ochrony (patrz Windows Experience Blog: dostępność Windows 11).

W praktyce Windows 11 wprowadził twardszą granicę: wymagany jest m.in. TPM 2.0, a lista wspieranych procesorów jest bardziej restrykcyjna niż w poprzednich migracjach. Microsoft argumentował tę decyzję koniecznością podniesienia poziomu bezpieczeństwa (ochrona tożsamości, kluczy szyfrujących, mechanizmów uwierzytelniania i integralności systemu), co miało znaczenie zwłaszcza w świecie masowych ataków, ransomware i ochrony danych na urządzeniach przenośnych (patrz Microsoft Support: TPM 2.0 i Windows 11; patrz Microsoft: wymagania Windows 11).

Jeżeli spojrzymy na cykl życia, Windows 11 jest rozwijany „wersjami" (np. 23H2, 24H2, 25H2), z konkretnymi datami końca wsparcia dla wydań Home i Pro. To ważne w praktyce: nie wystarczy „mieć Windows 11" – trzeba też być na wspieranej wersji, żeby otrzymywać aktualizacje zabezpieczeń. Przykładowo: 24H2 miało wsparcie do 13 października 2026, a 25H2 do 12 października 2027 (patrz Microsoft Lifecycle: Windows 11 Home and Pro). W mojej ocenie to dobry moment, żebyście zauważyli różnicę w myśleniu o systemie: kiedyś „kupowało się wersję", dziś zarządza się „ciągłym strumieniem aktualizacji" i jego konsekwencjami dla kompatybilności, bezpieczeństwa i administracji.

1.4. macOS

Mac OS (klasyczny) to historyczna rodzina systemów operacyjnych dla komputerów Macintosh firmy Apple. To właśnie Macintosh spopularyzował wśród szerokiej publiczności graficzny interfejs użytkownika (GUI) jako podstawowy sposób pracy z komputerem osobistym. Zauważcie jednak, że sama idea GUI nie narodziła się w próżni – część rozwiązań i inspiracji Apple zaczerpnęło z laboratoriów Xerox PARC w Palo Alto, gdzie w 1973 roku powstał komputer Alto, będący jednym z pionierskich przykładów GUI w praktyce. patrz https://en.wikipedia.org/wiki/Xerox_Alto

Pierwsza wersja systemu Mac OS weszła do użytku publicznego w 1984 roku wraz z modelem Macintosh - reklamowana przy pomocy klipu nawiązującego do literatury Georga Orwella. Upowszechnione przez nią takie standardy graficznego interfejsu jak okna, rozwijalne menu, kursor myszy, kosz na niepotrzebne pliki stały się od tej pory podstawą współczesnych systemów operacyjnych. Zachęcony sukcesem Apple, w tym samym kierunku zaczął podążać Microsoft, jednocześnie Apple starał się opatentować wygląd i wrażenia swojego systemu. Doprowadziło to do długotrwałego procesu o własność intelektualną.

Mac OS działał w systemach opartych początkowo na procesorach Motoroli z linii 680x0, a następnie linii PowerPC opracowanej wspólnie przez Apple, Motorolę i IBM. W praktyce oznaczało to, że przez długi czas Apple utrzymywało własny, dość odrębny ekosystem sprzętowy, a zgodność aplikacji była silnie powiązana z daną architekturą procesora.

Mimo, że interfejs klasycznego Mac OS był bardzo intuicyjny i wygodny, sam system cierpiał na wiele poważnych bolączek, jak np. brak wielozadaniowości z wywłaszczeniem czy w pełni bezpiecznej pamięci. Apple przez wiele lat starało się rozwiązać te kłopoty na bazie kodu Mac OS. Starania te nie przyniosły zadowalającego rezultatu. Projekt Copland został zarzucony, natomiast Rhapsody stał się rozwiązaniem przejściowym, które wyznaczyło kierunek w stronę systemu nowej generacji.

Po powrocie do Apple jednego z założycieli firmy Steve'a Jobsa i wchłonięciu jego firmy NeXT rozpoczęto prace nad nowym systemem - Mac OS X. Prowadzono je w oparciu o doświadczenia z Rhapsody oraz NextStep, rewolucyjnego systemu operacyjnego stworzonego w ramach projektu firmy NeXT utworzenia nowego komputera osobistego. W międzyczasie doskonalono Mac OS, aby nie utracić pozycji rynkowej. Jego ostatnia wersja była oznaczona numerem 9.2.2.

Mac OS X / OS X / macOS

System Mac OS X został zapowiedziany jako następna generacja systemu dla Macintosh i trafił do sprzedaży w marcu 2001 roku. patrz https://www.apple.com/newsroom/2001/01/09Apples-Mac-OS-X-to-Ship-on-March-24/

Mimo kolejnego numeru (po 9 wersji) Mac OS X był w praktyce nowym produktem. Jak wspomnieliśmy - system Mac OS X wyrósł z linii NextStep, a więc z tradycji uniksowej. W efekcie jest zgodny ze standardami POSIX i bywa traktowany jako kolejna mutacja rodziny Unix. Z czasem Apple skróciło nazwę do OS X, a w 2016 roku wprowadzono nazwę macOS, aby ujednolicić nazewnictwo z innymi systemami Apple. patrz https://www.wired.com/2016/06/apple-os-x-dead-long-live-macos



Wygląd przykładowego ekranu w pierwszym systemie systemie Mac OS X.

Jądro systemu macOS (XNU) jest jądrem hybrydowym: łączy elementy Mach z warstwą BSD i zestawem mechanizmów potrzebnych do praktycznej pracy systemu. W efekcie zachowano część idei mikrojądra, ale równocześnie wiele kluczowych usług działa blisko jądra, bo w systemie masowym wydajność i kompatybilność są głównymi wymaganiami. patrz https://en.wikipedia.org/wiki/XNU

Firma Apple wraz z wprowadzeniem Mac OS X zdecydowała się zmienić również graficzną powłokę na zupełnie nowe rozwiązanie, nazwane Aqua, która została wyjątkowo dobrze przyjęta przez rynek, głównie ze względu na jej estetyczne walory. Sam Steve Jobs podkreślał, że "*One of the design goals was when you saw it you wanted to lick it*", czyli w wolnym tłumaczeniu - jednym z celów projektu było doprowadzenia użytkownika do chęci polizania interfejsu kiedy go zobaczy. Powłoka Aqua jako rozwiązanie masowe wykorzystywała efekty graficzne udostępnione przez ówczesne karty - przeźroczystość, animacje, odbicia i tym podobne, przy czym pozostała wygodna dla użytkowników.

Warto też zauważyć, że macOS przechodził kilka dużych zmian sprzętowych. Najpierw Apple przeszło z PowerPC na procesory Intela, a następnie – od 2020 roku – rozpoczęło przejście na własne układy Apple silicon, które zakończono w skali całej linii produktowej w kolejnych latach. To przejście było ważne nie tylko marketingowo: zmieniło profil wydajność/zużycie energii, a równocześnie zbliżyło „świat Maca” do świata iPhone’a i iPada (choćby przez podobieństwo architektury). patrz https://www.apple.com/pl/newsroom/2020/06/apple-announces-mac-transition-to-apple-silicon/

Darwin, XNU i „uniksowy spód” systemu

Gdy patrzymy na macOS „od spodu”, warto rozdzielić dwie perspektywy: użytkową (interfejs, aplikacje, wygoda) oraz systemową (jądro, sterowniki, usługi). Ten drugi obszar wiąże się z warstwą Darwin i jądrem XNU, dzięki czemu macOS dziedziczy wiele cech systemów uniksowych: narzędzia tekstowe, model uprawnień, procesy i sygnały, oraz zgodność z częścią standardu POSIX. W praktyce oznacza to, że wiele narzędzi administracyjnych i deweloperskich znanych ze świata Unix/Linux ma na Macu naturalne odpowiedniki.

Jednocześnie trzeba pamiętać, że „zgodność z POSIX” nie oznacza „to samo co Linux”: macOS ma własne frameworki, własny model instalacji oprogramowania, inne podejście do sterowników i inny cykl rozwoju. To właśnie bywa dla czytających źródłem pierwszych niespodzianek: podobieństwo jest duże na poziomie pojęć, ale szczegóły implementacji potrafią się różnić.

Bezpieczeństwo: rola SIP i model współczesny

W nowszych wersjach macOS bezpieczeństwo stało się jednym z głównych filarów projektu systemu. Przykładem jest mechanizm System Integrity Protection (SIP), który ogranicza nawet konto root, jeśli chodzi o modyfikowanie wybranych, chronionych fragmentów systemu. Zauważcie, że to istotna zmiana mentalna: dawniej administrator (root) „mógł wszystko”, dziś system celowo odbiera część tej władzy, żeby utrudnić trwałe infekcje i sabotaż systemu przez złośliwe oprogramowanie. patrz https://support.apple.com/en-us/102149

W praktyce model bezpieczeństwa macOS opiera się na kilku warstwach naraz: uprawnieniach i izolacji procesów, podpisywaniu kodu, kontroli pochodzenia aplikacji (w tym mechanizmach weryfikacji), a także rozdziale tego, co może aplikacja, a co wymaga świadomej zgody użytkownika. Dla użytkownika jest to czasem uciążliwe, ale z punktu widzenia systemów produkcyjnych – zwłaszcza w firmach – takie „tarcie” bywa po prostu ceną za większą odporność na atak.

APFS: snapshoty i konsekwencje dla danych

W obszarze pracy z danymi ważną zmianą było wprowadzenie systemu plików APFS, zaprojektowanego pod współczesne nośniki i oczekiwania, takie jak szyfrowanie czy mechanizmy typu copy-on-write. Dla nas, praktyków, istotne są konsekwencje: snapshoty (migawki) ułatwiające odzysk danych po awarii lub błędzie, oraz naturalne wsparcie dla nowoczesnych mechanizmów bezpieczeństwa. patrz https://www.apple.com/pl/newsroom/2017/06/macos-high-sierra-delivers-advanced-technologies-for-storage-video-and-graphics/

To jest dobry moment, aby zapamiętać prostą obserwację: wiele „systemowych” technologii nie robi wrażenia na slajdach marketingowych, ale robi ogromną różnicę w codziennym administrowaniu i utrzymaniu komputerów. Jeśli system daje nam bezpieczniejsze podstawy przechowywania danych, to później mniej czasu spędzamy na gaszeniu pożarów.

Aktualna wersja systemu nosi nazwę macOS Tahoe i jest rozwijana w gałęzi 26.x (np. 26.2). patrz https://support.apple.com/en-us/122868

1.5. Podsumowanie

Wybór systemu operacyjnego powinien być ściśle związany z docelowym przeznaczeniem komputera, na którym ma być zainstalowany i wykorzystywany. Dobór powinien również uwzględniać administratora komputera oraz jego docelowego lub docelowych użytkowników w zakresie znajomości danego systemu, jak również ogólnej wiedzy i umiejętności posługiwania się oprogramowaniem. Inaczej powinien być dobrany system dla komputera domowego, serwera wydruków w małej firmie czy serwera przechowującego dane finansowe w banku. Decydując się na konkretny system operacyjny należy mieć świadomość o jego niezawodności i stabilności, możliwościach tworzenia kopii zapasowej itd.

Do użytku osobistego w znaczącej większości wykorzystuje się dziś systemy z rodziny Microsoft Windows (głównie Windows 10 i 11), macOS (na komputerach Apple) oraz Linux (w różnych dystrybucjach). Zdecydowanie najpopularniejszym rozwiązaniem pozostaje system Windows. W niektórych zastosowaniach nadal trudno go zastąpić (np. gry, część specjalizowanego oprogramowania firmowego oraz wybrane narzędzia inżynierskie, takie jak CAD). W wielu przypadkach jednak inne rozwiązania sprawdzą się tak samo dobrze albo i lepiej. Typowym przykładem jest tutaj przetwarzanie grafiki czy montaż wideo, gdzie macOS bywa częstym wyborem ze względu na spójność środowiska, stabilność oraz jakość ekosystemu aplikacji.

W zastosowaniach serwerowych częściej wykorzystuje się różne wersje systemów z rodziny Unix, włączając w to rozwiązania płatne (np. Red Hat Enterprise Linux, AIX) oraz darmowe (różne dystrybucje Linux-a, FreeBSD czy OpenBSD). Wynika to przede wszystkim z dojrzałego modelu usług sieciowych, łatwej automatyzacji, oraz wieloletniej praktyki administracyjnej tych systemów. Jednocześnie Windows ma własną, dojrzałą linię serwerową i w wielu organizacjach jest kluczowy ze względu na usługi katalogowe, domeny i integrację z ekosystemem Microsoft.

W ostatnich latach widać też stały rozwój gałęzi Linux w kontekście zastosowań osobistych: dystrybucje stały się łatwiejsze w instalacji i utrzymaniu, a środowiska graficzne oraz aplikacje są znacznie dojrzalsze niż kilkanaście lat temu. Nie oznacza to jednak rewolucji udziałów rynkowych na desktopie: wzrost jest raczej powolny i „krok po kroku”, ale dla wielu użytkowników to już pełnoprawna alternatywa.

W praktyce ciężko jest zbadać rzeczywisty udział poszczególnych systemów operacyjnych na rynku, bo zależy on od tego, co dokładnie mierzymy: instalacje na komputerach, sprzedaż urządzeń, czy ruch internetowy. Dla komputerów stacjonarnych i laptopów (desktop) nadal dominuje Windows – zarówno w Polsce, jak i globalnie – ale macOS i Linux mają dziś zauważalny udział. Przykładowo, w pomiarach dla desktopu w Polsce (styczeń 2026) Windows ma około 80%, a macOS i Linux po kilka procent. Globalnie dla desktopu Windows ma około 67–68%, a Linux około 4%. patrz https://gs.statcounter.com/os-market-share/desktop/poland oraz https://gs.statcounter.com/os-market-share/desktop/worldwide

W tym miejscu muszę dopowiedzieć coś o systemach mobilnych, bo one realnie „przejęły” dużą część codziennego korzystania z Internetu. Jeśli w statystykach obejmiemy wszystkie urządzenia (a nie tylko komputery), dominują dziś systemy smartfonowe – przede wszystkim Android oraz iOS. W wielu badaniach ruchu sieciowego to właśnie urządzenia mobilne odpowiadają za znaczną część odsłon i czasu spędzanego w sieci. patrz https://ranking.gemius.com/pl/ranking/systems/

Dlaczego w tej lekcji pisaliśmy o nich mniej? Bo klasyczny podział, który tu omawiamy (desktop ↔ serwer), dotyczy przede wszystkim komputerów ogólnego przeznaczenia i usług sieciowych. Systemy mobilne są projektowane pod inne priorytety: energooszczędność, stałą pracę w sieciach radiowych, bardzo restrykcyjny model uprawnień aplikacji (sandbox), oraz dystrybucję oprogramowania przez sklepy aplikacji. To w praktyce inna szkoła projektowania systemu i inny zestaw problemów.

Warto jednak zauważyć powiązania: Android opiera się na jądrze Linux, a iOS wywodzi się z tej samej rodziny technologii co macOS (Darwin/XNU). Różnica polega na tym, że mobilne systemy są znacznie bardziej „zamknięte” operacyjnie: aplikacje mają mniej swobody, a system mocniej wymusza standardy bezpieczeństwa i prywatności.

Spis treści

1. Bezpieczeństwo w systemach informatycznych
1.1. Ochrona
1.2. Bezpieczeństwo
1.3. Kryptografia
1.4. Kerberos
1.5. Podsumowanie

1. Bezpieczeństwo w systemach informatycznych

W tej lekcji chciałbym się skupić na zagadnieniach ogólnie rozumianego bezpieczeństwa w systemach informatycznych. Ze względu na fakt, iż niniejszy podręcznik i przedmiot ma stanowić wprowadzenie w zagadnienia informatyki - nie znajdziecie tu jednak kompendium wiedzy o ochronie systemów komputerowych. Raczej postaram się przedstawić Wam podstawowe pojęcia związane z ochroną i bezpieczeństwem systemów informatycznych, oraz omówić proste metody zapobiegania zagrożeniom. Tak więc - znajdziecie tu raczej przyczółek do dalszych, samodzielnych poszukiwań, niż szczegółowy, prowadzący krok po kroku, kurs pt. "Jak zabezpieczyć mojego PC".

Bezpieczeństwo danych i systemów

Zagadnienia bezpieczeństwa systemów informatycznych obejmują całe, dość szerokie spektrum tematów. Wychodzę tu z założenia, że system operacyjny wchodzi w skład większego organizmu jakim jest *system informatyczny*. Pod pojęciem **systemu informatycznego** rozumiem zespół elementów sprzętowych i programowych służący do gromadzenia, przetwarzania i przesyłania danych.

To, co zazwyczaj chcemy zabezpieczyć - to wspomniane dane. Nie będę się tutaj zastanawiał co oznacza określenie "Kensington Lock" oraz jakim łańcuchem i do czego przymocować komputer ... ;) (patrz Komputery przenośne HP – Używanie blokady Kensington).

Problem bezpieczeństwa i ochrony danych (informacji) przechowywanych przez systemy informatyczne nabiera coraz większego znaczenia. Liczba osób korzystających z systemów informatycznych ciągle rośnie, a ostatnie lata przynoszą ciągły, dynamiczny rozwój usług internetowych, usług teleinformatycznych, handlu elektronicznego i bankowości elektronicznej. Łatwo sobie wyobrazić jakie skutki dla funkcjonowania przedsiębiorstwa mogą wiązać się z utratą lub niekontrolowaną zmianą danych elektronicznych. Często też (niestety coraz częściej) słyszycie o udanych atakach na komputery / serwery / sieci komputerowe, o kradzieży danych, kont, czy też pieniędzy. Nie wynika to wyłącznie ze słabości systemów zabezpieczeń (te bywają coraz lepsze i bardziej wyrafinowane) - lecz także ze skali stosowania systemów komputerowych i automatyzacji ataków. W zasadzie każdy komputer podłączony do internetu podlega nieustannym próbom ataku różnych typów. Teraz chciałbym się skupić na zagrożeniach dla systemów komputerowych.

Zagrożenia

Zagrożenia dla danych przechowywanych w systemach teleinformatycznych są bardzo różnorodne. Należą do nich zarówno działania mogące spowodować zniszczenie, uszkodzenie lub zmianę danych w systemie jak również zmierzające do odczytania (skradzenia) ich przez osoby nieuprawnione.

Zbiory danych należące do użytkownika podłączonego do systemu otwartego (a więc udostępnionego w sieci publicznej) mogą zostać zniszczone np. w wyniku działania złośliwego oprogramowania lub odczytane przez osobę atakującą. Mogą też zostać zniekształcone lub zniszczone w czasie transmisji np. w wyniku awarii lub też celowego działania innych osób. W ich wyniku może nastąpić zarówno utrata danych, przechwycenie przez osoby nieupoważnione jak również naruszenie integralności i spójności systemu (bazy danych), do którego dane te zostały przesłane. Dlatego też specjaliści dzielą istniejące zagrożenia na kilka grup:

- Włamania i kradzież danych i oprogramowania. Tego typu zagrożenia pojawiły się stosunkowo dawno, i zazwyczaj opierają się na przejęciu kontroli nad atakowanym systemem. Po dokonaniu włamania napastnicy mogą wykorzystać zaatakowany system w zasadzie dowolnie - tzn przejąć zawartość jego dysków twardych wraz z znajdującymi się tam danymi wrażliwymi i / lub oprogramowaniem, podszywać się pod prawowitego właściciela danego systemu, czy też wykorzystać go do dalszych ataków na inne systemy. W praktyce „wejście” do systemu bywa realizowane nie tylko przez luki techniczne, ale też przez przejęte poświadczenia, błędy konfiguracji lub socjotechnikę (np. phishing).
- Oszustwa i fałszerstwa. Zagrożenie spotykane bardzo często w sieci rozległej - polega na próbie wyłudzenia informacji za pomocą spreparowanych maili, stron, itp. Jak zapewne się domyślacie, poziom wiedzy technicznej wymaganej do przeprowadzenia takiego ataku bywa zdecydowanie niższy niż w przypadku włamań opartych o luki w oprogramowaniu. Jednocześnie, mimo „prostoty”, jest to wektor realnie skuteczny: w raportach o naruszeniach regularnie pojawia się problem kradzieży poświadczeń i phishingu (patrz Verizon Data Breach Investigations Report (DBIR)).
- Szpiegostwo (podsłuch, kradzież danych). W odróżnieniu od oszustw - w przypadku szpiegostwa mówimy o działaniu typowo pasywnym, tzn. podsłuchiwaniu sieci albo na poziomie fizycznym, albo na poziomie wyższych warstw, lecz bez ingerencji ani w podsłuchiwane systemy, ani w zawartość przesyłanej informacji
- Niszczenie danych. Jest to działanie destrukcyjne, zorientowane na doprowadzenie do strat u właściciela atakowanego systemu.
- Paraliżowanie pracy systemów komputerowych. Dość częste ataki, polegające na uniemożliwianiu normalnej pracy systemów, szczególnie serwerów w internecie, bez ingerencji w spójność przechowywanych na nich danych, oraz bez prób przejmowania nad nimi kontroli. Cała złośliwość ataku sprowadza się do uniemożliwienia skorzystania z określonej usługi przez podmioty uprawnione.
- Fałszowanie i uniemożliwianie transmisji danych. To zagrożenie jest raczej elementem wojny elektronicznej niż zagrożeniem spotykanym na co dzień w sieci.

Jak widać, jest się czego bać ... skoro tak, to zanim przejdziemy do dokładniejszego omawiania zagadnienia, wprowadźmy kilka pojęć. Zazwyczaj ochronie podlegają dane - ale co to oznacza? W nomenklaturze specjalistycznej stosuje się różne cele ochrony, tzn:

- poufność (ang. confidentiality) - ochrona informacji przed nieautoryzowanym jej ujawnieniem. Tak zabezpieczacie zazwyczaj swoje prywatne dane na dysku - zależy Wam na tym by nikt niepowołany nie był w stanie ich odczytać.
- integralność danych (data integrity) - ochrona informacji przed nieautoryzowanym jej zmodyfikowaniem. Tak zabezpiecza się zawartość publicznych stron www - tam informacja jest dostępna dla każdego, lecz tylko uprawnieni mogą ją modyfikować.

- dostępność (ang. availability) - zapewnienie, że uprawnieni użytkownicy i systemy mają dostęp do danych i usług wtedy, gdy są potrzebne; ten cel ochrony jest bezpośrednio związany z atakami DoS/DDoS (patrz NIST SP 1800-26 (CIA triad)).
- autentyczność (ang. authenticity) - pewność co do pochodzenia, autorstwa i treści danych. Temu służy podpis elektroniczny, lub system certyfikatów serwera.
- niezaprzeczalność (ang. nonrepudiation) - ochrona przed fałszywym zaprzeczeniem przez nadawcę – faktu wysłania danych, przez odbiorcę - faktu otrzymania danych. W praktyce bywa trudna do zapewnienia.

Skoro już jesteśmy przy nomenklaturze, to jeszcze pozwolicie, iż nazwiemy fachowo najczęściej spotykane zagrożenia:

- **hacking** włamywanie do systemów lub sieci komputerowych; w zależności od okoliczności może stanowić czyn zabroniony w rozumieniu art. 267 Kodeksu karnego (patrz Kodeks karny (ISAP)).
- **sniffing** podsłuchiwanie pakietów w sieci; w zależności od sposobu i celu działania może wiązać się z nieuprawnionym uzyskaniem informacji lub dostępu do systemu (patrz Kodeks karny (ISAP)).
- **session hijacking** polega na uzyskiwaniu nieuprawnionego dostępu do systemów poprzez przechwycenie sesji legalnego użytkownika
- **Spoofing** podszywanie się pod innego użytkownika lub system w sieci.
    - **IP spoofing** - rodzaj ataku w którym adres IP napastnika jest przedstawiany jako adres zaufany.
    - **TCP spoofing** - podszywanie bazujące na przejęciu lub odtworzeniu parametrów połączenia (np. numerów sekwencyjnych) w celu „udawania" innej strony połączenia.
    - **UDP spoofing** – podszywanie się i/lub modyfikacja pakietów w celu atakowania usług i protokołów wykorzystujących protokół UDP.
    - **ARP spoofing** - rodzaj ataku w którym adres MAC napastnika jest przedstawiany jako adres zaufany (np. poprzez wysyłanie fałszywych odpowiedzi na pytania ARP).
    - **DNS spoofing** – rodzaj ataku polegający na podaniu ofierze fałszywej odpowiedzi DNS (np. przez zatruwanie cache lub podszycie odpowiedzi), w wyniku czego ofiara trafia na nieprawidłowy adres IP.

- **phishing** (password fishing) - metoda uzyskiwania haseł dostępu do zasobów internetowych (kont bankowych użytkownika) za pośrednictwem e-maili i fałszywych stron. Atakujący podszywa się pod przedstawiciela instytucji finansowej próbując nakłonić odbiorców e-maili do podania haseł oraz innych danych osobowych (np. poprzez fałszowanie stron WWW).
- **cracking** - łamanie zabezpieczeń programów komputerowych, dostępu do systemów, usług sieciowych.
- **piractwo komputerowe** - kopiowanie, reprodukowanie, używanie i wytwarzanie bez zezwolenia produktu chronionego przez prawo autorskie.

Sporo, prawda? A to tylko subiektywny wybór autora tego podręcznika spośród mrowia innych nazw i terminów, zogniskowany na zagrożeniach prowadzących do włamania do systemu komputerowego. Przy czym, jak wspominałem - do włamywania się do systemów niezbędna jest wiedza - zarówno ta techniczna, z dziedziny informatyki, jak też trochę psychologii ... dlaczego psychologii? Bo ogromna część realnych incydentów zaczyna się od przejęcia poświadczeń (hasła, tokeny, sesje) albo od skłonienia użytkownika do wykonania niebezpiecznej czynności. W raportach dotyczących naruszeń regularnie pojawia się np. problem „stolen credentials" (patrz Verizon DBIR).

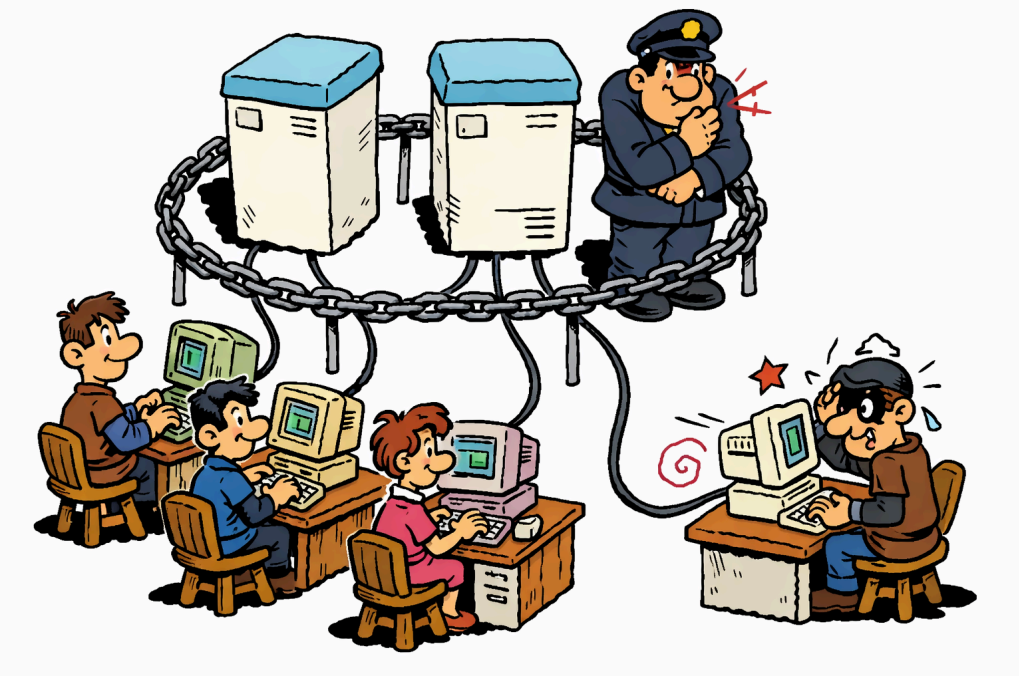
Współcześnie najczęściej spotykane zagrozenia to:

- **Ransomware i wymuszenia (często „podwójne" lub „wielokrotne").** Atakujący szyfruje dane lub blokuje systemy, a równolegle grozi ujawnieniem wykradzionych informacji. To dziś jeden z kluczowych problemów bezpieczeństwa organizacji (patrz CISA: StopRansomware – Ransomware Guide oraz ENISA: Threat Landscape 2024).
- **Kradzież poświadczeń i przejęcia kont.** W praktyce wiele incydentów zaczyna się od wyłudzenia hasła (phishing / pretexting), użycia wykradzionych baz haseł (credential stuffing) albo od malware wykradającego loginy i tokeny. To „cichy" wektor ataku, bo napastnik loguje się jak prawidłowy użytkownik (patrz Verizon: Data Breach Investigations Report (DBIR)).
- **Phishing i socjotechnika.** To nie tylko „maile od banku", ale też fałszywe prośby o dopłatę, dokumenty w chmurze, rozmowy telefoniczne, fałszywe wsparcie techniczne i podszywanie się pod przełożonych lub kontrahentów. W wielu raportach jest to stały, powtarzający się mechanizm wejścia do organizacji (patrz Verizon DBIR).
- **Ataki na aplikacje webowe i błędy logiki autoryzacji.** W praktyce chodzi o luki w kodzie i konfiguracji (np. błędna kontrola dostępu, wstrzyknięcia, błędy uwierzytelniania, błędy w API). To typowa przyczyna wycieków danych i przejęć usług (patrz OWASP Top Ten).
- **Błędne konfiguracje i nadużycia środowisk chmurowych.** Współcześnie dużo usług stoi w chmurze, a część incydentów wynika z błędów konfiguracji (np. zbyt szerokie uprawnienia, źle wystawione zasoby, klucze i sekrety w repozytoriach) oraz z braku kontroli nad dostępem (patrz ENISA: Threat Landscape 2024).
- **Ataki na łańcuch dostaw.** Zamiast atakować jedną firmę, napastnik kompromituje dostawcę (oprogramowania, usług, integratora), komponent lub bibliotekę i „wchodzi" dalej przez zaufane zależności. To trudne do wykrycia i bardzo kosztowne (patrz ENISA: Finance Threat Landscape (sekcja supply chain)).
- **Ataki na dostępność usług (DoS/DDoS).** Wciąż aktualne: celem jest przeciążenie usług i uniemożliwienie działania (np. serwisów WWW, API, usług krytycznych). W praktyce często są to ataki z wielu urządzeń (botnet) (patrz CISA: Understanding and Responding to DDoS Attacks).

Sporo tych zagrożeń ... jak się przed nimi chronić? No cóż, w 100% zabezpieczyć się nie da, ale zawsze coś można zrobić ;)

1.1. Ochrona

Ochrona jest problemem wewnętrznym systemu informatycznego i dotyczy mechanizmów służących do kontrolowania dostępu programów, procesów lub użytkowników do zasobów zdefiniowanych w systemie komputerowym. Zauważcie - iż na poziomie systemu mówi się o dostępie nie tylko do danych, a właśnie zasobów, czyli pojęcia szerszego - zasobem jest też drukarka, pamięć operacyjna, itp.



Cele ochrony

Pierwsze systemy komputerowe wyposażone były w bardzo słabe mechanizmy ochrony (w zasadzie można stwierdzić, iż nie było ich wcale). Wraz z pojawieniem się systemów wieloprogramowych problem ochrony zaczął nabierać istotnego znaczenia i stał się bardziej intensywnym obiektem zainteresowań twórców systemów operacyjnych, i szerzej rozumianych systemów informatycznych. Pewne wstępne informacje o ochronie zasobów przez system już Wam przedstawiłem w trakcie omawiania systemów operacyjnych - tutaj spojrzymy na to zagadnienie z nieco innego punktu widzenia.

Po pierwsze - większość współczesnych systemów komputerowych pracuje w sieci. Tak więc - ich ochrona rozciąga się także na zasoby rozproszone po sieci. Fachowo mówi się, iż prawidłowa ochrona w systemie dostarcza odpowiednich mechanizmów, które mogą posłużyć do realizacji określonej polityki wykorzystania zasobów systemu, definiowanej zarówno na poziomie lokalnym, jak i zdalnym. Przy czym poprzez politykę wykorzystania zasobów w uproszczeniu możecie rozumieć możliwość pełnej kontroli nad dostępem do nich, odczytem, zapisem, oraz wykonaniem.

Do mechanizmów ochrony należą między innymi:

- bezpieczne dzielenie wspólnej przestrzeni nazw logicznych, np. katalogów plików;
- bezpieczne dzielenie wspólnej przestrzeni obiektów fizycznych, np. pamięci.

Podstawowym celem stosowania ochrony jest zapewnienie, aby każdy wykonywany proces korzystał tylko i wyłącznie z zasobów mu przyznanych i to w sposób określony przez politykę ochrony. Samo definiowanie polityki może być realizowane w różny sposób. W lekcji poświęconej plikom omawialiśmy zasady i prawa dostępu do nich - to właśnie, w połączeniu z tworzeniem grup użytkowników i przydzielaniem ich do nich - jest prostą realizacją zapisu polityki dostępu do plików na dysku. W praktyce, w skomplikowanych, rozproszonych systemach informatycznych stosuje się daleko bardziej skomplikowane mechanizmy ochrony.

Modele ochrony

W klasycznych rozważaniach o systemach operacyjnych często spotkacie domenowy model ochrony zasobów. Jest to wygodny model pojęciowy, który pozwala opisać: kto (użytkownik albo proces) ma jakie prawa do jakich obiektów (plików, urządzeń, pamięci, itp.). W praktyce współczesnych systemów spotyka się też podejścia oparte o role i atrybuty (np. RBAC i ABAC) oraz rozwiązania zarządzania tożsamością i dostępem w środowiskach rozproszonych (IAM) (patrz NIST SP 800-162 (ABAC)).

Definicja

Domena ochrony jest zbiorem obiektów i rodzajów operacji, które można wykonać dla danego obiektu.

Obiektem może być wszystko to co system operacyjny obsługuje i udostępnia: segment pamięci, procesor, drukarka, dysk, plik, program itd. W zależności od rodzaju z danym obiektem związane są określone operacje. Na przykład z plikiem związane są takie operacje jak: otwarcie, zamknięcie, odczyt, zapis (modyfikacja), usunięcie czy wykonanie, a z urządzeniem USB - możliwość zamontowania, odmontowania, odczytu jakichkolwiek danych, ich zapisu czy wykonania programów z nośnika.

W modelu domenowym korzysta się również z pojęcia prawa dostępu - nadawanego użytkownikowi systemu, zarówno jeśli tym użytkownikiem jest człowiek, jak i inny system informatyczny czy też uruchomiony program (proces) w systemie. Prawo dostępu definiuje nam zbiór operacji, które można wykonać na określonym obiekcie. W prawidłowo chronionym systemie trzymamy się założenia, iż nigdy nikt nie powinien mieć większych praw niż są one mu niezbędne do realizacji swoich zadań. Fachowo takie podejście nazywamy zasadą najmniejszych uprawnień.

Definicja

Zasada najmniejszych uprawnień (*ang. least privilege*) mówi, że użytkownik lub proces w systemie ma tylko takie uprawnienia i dostęp do takich zasobów, które są mu niezbędnie potrzebne do wykonania zadania (patrz NIST Glossary: Least Privilege).

W praktyce spotkacie też określenie „zasada wiedzy koniecznej” (*need-to-know*), które częściej podkreśla aspekt dostępu do informacji: dostęp powinien mieć ten, kto rzeczywiście potrzebuje danej informacji, a nie „kto tylko może”. W prostych systemach te dwa podejścia wyglądają podobnie, ale w większych organizacjach warto pamiętać o tym rozróżnieniu.

Ścisłe trzymanie się tej zasady pozwala na minimalizację szkód - np. w wyniku włamania na konto użytkownik może wykonać jedynie te operacje, co do których ma prawo wykonania, w przypadku przejęcia kontroli nad procesem możliwe jest wykonanie jedynie tych operacji, do których uprawniony jest dany proces, itp...

Zazwyczaj domena ochrony jest w systemach informatycznych powiązana z użytkownikiem danego systemu - i tak, uruchomiony program ma takie prawa jak użytkownik który go uruchamiał. Jeśli więc użytkownik nie ma prawa do zmiany zawartości katalogu systemowego, to żaden program uruchomiony przez tego użytkownika zmian w katalogach systemowych nie wykona.

Niestety, zasada najmniejszych uprawnień w przeszłości nazbyt często była ignorowana. Przez długi czas w systemach z rodziny Windows wielu użytkowników pracowało na kontach o szerokich uprawnieniach. W nowszych wersjach Windows stosuje się mechanizmy ograniczające to ryzyko, w szczególności UAC (User Account Control), które wymusza „podniesienie” uprawnień dopiero wtedy, gdy jest to potrzebne (patrz Microsoft: How User Account Control works).

W rozwiązaniach Unix-owych i pochodnych (np. Linux) wyraźnie rozróżnia się konto administratora (*root*) i konta zwykłych użytkowników. Technicznie da się pracować jako root, ale nie jest to zalecane: standardem jest praca na koncie zwykłym i podnoszenie uprawnień tylko na czas wykonania konkretnej czynności administracyjnej (np. poleceniem *sudo*).

Jeszcze dalej ochrona jest rozwinięta w systemach dla urządzeń przenośnych (Android, iOS) - tam prawa są nadawane aplikacjom, a system dba o ich izolację (np. przez mechanizmy sandbox) oraz o kontrolę uprawnień do zasobów i danych (patrz Android Open Source Project: Application Sandbox).

1.2. Bezpieczeństwo

Bezpieczeństwo jest zagadnieniem szerszym niż ochrona systemu. Ochrona dotyczy przede wszystkim mechanizmów kontroli dostępu wewnątrz systemu, natomiast bezpieczeństwo obejmuje także odporność na ataki z zewnątrz, błędy i nadużycia wewnętrzne, awarie oraz zdarzenia losowe. Uważam, że system jest bezpieczny wtedy, gdy potrafi ograniczać skutki zagrożeń (np. prób naruszenia poufności danych), a jednocześnie umożliwia wykrycie incydentu i odtworzenie poprawnego działania.

Polityka bezpieczeństwa

Bezpieczeństwo systemu zależy w dużej mierze od przyjętej polityki bezpieczeństwa. Nawet najlepsze zabezpieczenia nie uchronią przed kradzieżą lub zniszczeniem danych, jeśli polityka bezpieczeństwa pozwala na swobodny dostęp do konsoli administratora. Z tego punktu widzenia, polityka bezpieczeństwa, rozumiana jako zbiór wszystkich działań przyjętych w celu osiągnięcia wysokiego poziomu bezpieczeństwa systemu informatycznego, jest zagadnieniem kluczowym, i złożonym. Podstawowymi mechanizmami stosowanymi w celu zapewnienia bezpieczeństwa w systemie jest dokonywanie identyfikacji, uwierzytelniania oraz autoryzacji użytkowników systemu.

W praktyce równie ważna jest rozliczalność działań, czyli możliwość odpowiedzi na pytania: kto, kiedy i co zrobił. Do tego potrzebne są logi i audyt (rejestracja zdarzeń), bo bez nich trudno wykryć naruszenie, wyjaśnić incydent i wyciągnąć wnioski.

Identyfikacja

Identyfikacja polega na stwierdzeniu tożsamości użytkownika lub innego obiektu zamierzającego skorzystać z zasobów systemu. Najczęściej każdy użytkownik systemu ma przypisany unikalny w obrębie systemu identyfikator lub numer. Dzięki stosowaniu identyfikatorów system wie z jakim użytkownikiem ma do czynienia. Może również dokonywać rejestracji działań wykonanych przez tego użytkownika i na tej podstawie przedstawiać raporty lub dokonywać rozliczeń.

Uwierzytelnianie

Sama identyfikacja nie może jednak zapewnić bezpieczeństwa systemu. Identyfikatory użytkowników są najczęściej tworzone na podstawie dostępnego algorytmu i są wobec tego znane lub łatwe do odgadnięcia. Z tego powodu poza procesem identyfikacji przeprowadzany jest proces uwierzytelniania. Uwierzytelnianie polega na sprawdzeniu, że obiekt, który zgłasza żądanie dostępu do systemu jest tym za kogo się podaje.

Hasła

Najczęstszym sposobem uwierzytelniania użytkowników systemu jest wymaganie podania hasła. Długość i rodzaj hasła są oczywiście zależne od systemu oraz od przyjętej polityki bezpieczeństwa. Dawniej spotykało się też systemy przyjmujące bardzo krótkie hasła (np. do 8 znaków) albo słabo wspierające różne zestawy znaków - dziś traktuje się to raczej jako ograniczenie systemów legacy. Warto jest przestrzegać następujących ogólnych zasad (patrz NIST SP 800-63B):

- hasło powinno być długie; w praktyce bardzo dobrze sprawdzają się dłuższe frazy (tzw. passphrase);
- hasło powinno być unikalne dla każdego serwisu i systemu (nie używajcie tego samego hasła w wielu miejscach);
- nie używajcie haseł oczywistych lub łatwych do odgadnięcia (słownikowych, imion, dat urodzenia, nazw zespołów, itp.);
- jeśli macie wiele kont, rozważcie użycie menedżera haseł - pozwala on bezpiecznie przechowywać długie i unikalne hasła;
- nie należy zapisywać hasła w łatwo dostępnym miejscu, a już na pewno nie należy umieszczać go na obudowie komputera (!)

W praktyce dość silne, i jednocześnie łatwe do zapamiętania hasła da się tworzyć korzystając z pewnej reguły oraz zdań / stwierdzeń które są dla nas na tyle istotne, że je pamiętamy. Typowy przykład: Lom2yead. Na pierwszy rzut oka nie ma to sensu - ale to początek Pana Tadeusza: pierwsze litery z "Litwo ojczyzno moja", drugie z "ty jesteś jak zdrowie". Finalnie otrzymujemy hasło o statystycznej charakterystyce zbliżonej do rozkładu losowego, jednocześnie nie występujące w słownikach. Zauważcie jednak, że schematy mają wadę: jeśli ktoś pozna regułę, to może łatwiej odtworzyć podobne hasła.... Innym rozwiązaniem są bardzo długie, unikalne hasła lub passphrase. No i nie zapominajcie że najlepsze są czysto losowe ciągi, a trudności z ich pamiętaniem rozwiązuje menedżer haseł (patrz NIST SP 800-63B).

Ze względu na swoją naturalność i prostotę systemy identyfikacji opierające swoje działanie na hasłach są wciąż często stosowane w dzisiejszych systemach komputerowych. Stosowanie haseł w celu określenia tożsamości posiada jednak wiele wad. Najważniejsze z nich to:

- komputer musi przechowywać w pewnej postaci dane służące do weryfikacji haseł użytkowników (np. skróty/hashy);
- hasło może zostać przechwycone w czasie przesyłania go do komputera (dotyczy to szczególnie systemów uwierzytelniania w sieci). Dlatego też opracowano mechanizmy, które nie wymagają przesyłania hasła wprost lub ograniczają sens jego przechwycenia (np. protokoły wyzwanie-odpowiedź; w dalszej części omawiamy Kerberos);
- użytkownicy zapominają swoje hasła;
- użytkownicy wybierają hasła, które łatwo odgadnąć. Elementem przeciwdziałającym może być wymuszanie minimalnej długości i blokowanie typowych haseł oraz ograniczanie liczby prób logowania (patrz NIST SP 800-63B);
- użytkownicy ujawniają swoje hasła innym osobom.

Przedmioty

Oprócz lub obok uwierzytelniania za pomocą hasła stosuje się też uwierzytelnianie za pomocą specjalnych identyfikatorów (kart chipowych lub tokenów). Takie systemy również mają swoje wady. Przedmioty używane do identyfikacji w rzeczywistości nie potwierdzają tożsamości danej osoby, lecz posiadanie przedmiotu. Każdy, kto zdobędzie taki przedmiot będzie mógł podszywać się pod jego właściciela. Z tego też powodu takie systemy są często elementem uwierzytelniania wieloskładnikowego (MFA), czyli łączenia co najmniej dwóch metod uwierzytelniania.

Do tej samej grupy zaliczymy też systemy oparte na jednorazowych kodach przesyłanych na telefon komórkowy (SMS). W praktyce są one wygodne, ale nie zawsze najbezpieczniejsze (np. istnieją ataki typu przejęcie numeru/SIM-swap), dlatego tam gdzie to możliwe lepsze bywają aplikacje generujące kody jednorazowe, powiadomienia push lub sprzętowe klucze bezpieczeństwa (patrz NIST: Authenticators).

Indywidualne cechy użytkownika

Do uwierzytelniania stosowane są też metody biometryczne wykorzystujące fakt unikatowości pewnych cech fizycznych człowieka – takich jak linie papilarne, czy rysunek tęczówki oka lub też cech behawioralnych takich jak sposób mówienia, czy pisania.

Typowym przykładem uwierzytelniania na podstawie cech fizycznych użytkownika jest skanowanie linii papilarnych. Rozpoznawanie tej cechy dowiodło już swoją skuteczność, wiarygodność i wygodę. Skanowanie obrazu odcisku palca zabiera mało czasu i wysiłków użytkownika oraz jest jedną z najmniej inwazyjnych metod biometrycznych. Weryfikacja odcisku linii papilarnych jest również stosunkowo szybka, choć nie jest stuprocentowo bezpieczna.

> ### Komentarz
>
> W chwili obecnej weryfikacja użytkowników za pomocą ich cech fizycznych lub behawioralnych jest bardzo popularna. Część laptopów jest standardowo wyposażana w czytniki linii papilarnych. Technika ta bywa używana razem z weryfikacją opartą na haśle, a więc w formie uwierzytelniania wieloskładnikowego. Taki proces uwierzytelniania zwykle zapewnia wyższy poziom bezpieczeństwa, niż każda z tych metod oddzielnie. Natomiast przestrzegam przed bezwarunkową wiarą w odporność na oszukanie tanich skanerów linii papilarnych.

Autoryzacja

Po przeprowadzeniu identyfikacji i uwierzytelnienia obiektu system przydziela danemu obiektowi określone prawa dostępu do obiektów systemu informatycznego - ten etap jest zazwyczaj nazywany autoryzacją.

Zagrożenia bezpieczeństwa

Zagrożenia bezpieczeństwa systemu można podzielić na dwie zasadnicze grupy. Pierwsza grupa, to zagrożenia wynikające ze świadomego działania człowieka. Druga, to nieświadome działania użytkowników, awarie sprzętu, zaniki zasilania oraz oddziaływanie czynników zewnętrznych (pożary, powodzie, katastrofy).

Użytkownicy

Najczęstszym zagrożeniem dla systemu informatycznego jest działanie czynnika ludzkiego, a w tym użytkowników systemu. Z badań wynika, że wiele naruszeń bezpieczeństwa systemu jest wynikiem działań wewnątrz tego systemu (organizacji). Naruszenia te są zarówno wynikiem działań świadomych, jak też zwykłej nieuwagi czy też niedbalstwa użytkowników. Do najistotniejszych zagrożeń wewnętrznych należy zaliczyć:

- sabotaż wewnętrzny;
- kradzież informacji;
- kradzież usług;
- błędy użytkowników;
- niedbalstwo;
- nieprawidłowe stosowanie procedur polityki bezpieczeństwa.

Ataki na system

Atak na system przeprowadzany jest najczęściej poprzez przechwycenie danych uwierzytelniających użytkownika (hasła, tokeny, sesje) lub skłonienie go do wykonania niebezpiecznej czynności (np. phishing).

Do najczęstszych metod ataku należą:

- Włamanie (ang. *hacking*) do systemów lub sieci komputerowych (w zależności od okoliczności może stanowić czyn zabroniony w rozumieniu art. 267 Kodeksu Karnego; patrz Kodeks karny (ISAP)). Włamanie może następować za pomocą najróżniejszych dróg i technik, zarówno korzystając z luk w oprogramowaniu systemowym i jego zabezpieczeniach, jak i z naiwności użytkowników. Finalnym efektem włamania często jest przejęcie kontroli nad atakowanym systemem. Odbywa się to najczęściej albo poprzez nieautoryzowane zainstalowanie na nim złośliwego oprogramowania, albo poprzez zdobycie informacji umożliwiającej prawidłowe uwierzytelnienie się w systemie (zdobycie haseł), albo poprzez przejęcie sesji. W przypadku instalacji złośliwego oprogramowania, zazwyczaj dzieli się je na kilka grup:

    - **Wirusy**, czyli programy, które rozprzestrzeniają się poprzez infekowanie innych plików kopią samego siebie. Do plików, które mogą zostać zainfekowane należą binarne pliki programowe, oraz pliki dokumentów dla aplikacji, które zawierają obsługę różnych makrojęzyków na tyle zaawansowanych, by umożliwić działanie wirusa. W przeszłości część wirusów infekowała również sektory startowe dysku (MBR).
    - **Robale** - czyli programy komputerowe, które szerzą się przez wysyłanie swoich kopii do innych komputerów, które uruchamiają go i dalej rozprzestrzeniają do innych komputerów. Najczęściej robale rozprzestrzeniały się za pomocą programów pocztowych.
      Konie trojańskie, programy, które rzekomo mają wykonać dla nas coś pożytecznego lub interesującego, a w rzeczywistości przeprowadzają w tle złośliwe działania, podczas gdy użytkownik posługuje się głównym programem.
    - **Bomby logiczne,** złośliwe fragmenty kodu wstawione w normalne pod pozostałymi względami programy. Często są tam umieszczone przez autora programu lub osobę, która go współtworzy. Bomby logiczne mogą być ustawione tak, aby zaczęły działać w określonej porze, lub dniu.
    - **Ransomware** (szyfrowanie danych i wymuszenia). Jest to obecnie jedna z najbardziej dotkliwych klas ataków na organizacje i użytkowników. W typowym scenariuszu napastnik dostaje się do systemu (np. przez phishing, przejęte hasło, podatność w usłudze zdalnego dostępu albo błędną konfigurację), następnie podnosi uprawnienia, rozpoznaje sieć i zasoby (serwery plików, kopie zapasowe, kontrolery domeny), po czym uruchamia szyfrowanie plików. Coraz częściej spotyka się wariant „podwójnego wymuszenia": zanim dane zostaną zaszyfrowane, są też wykradane, a następnie napastnik grozi ich publikacją. Skutki bywają dwojakie: przestój (brak dostępności usług i danych) oraz ryzyko ujawnienia informacji.
      Ochrona przed ransomware nie sprowadza się do jednej „magicznej" aplikacji. Najlepiej działa podejście warstwowe, w którym kilka prostych zasad ogranicza zarówno ryzyko infekcji, jak i skutki ataku (patrz CISA: Ransomware Guide):
      Kopie zapasowe - to najważniejszy element obrony, bo umożliwia powrót do działania bez płacenia okupu. Kopie powinny być wykonywane regularnie, przechowywane także poza komputerem/serwerem (np. offline lub w repozytorium odpornym na modyfikacje), i okresowo testowane pod kątem odtwarzania.
      Aktualizacje - łatki bezpieczeństwa systemu i aplikacji zmniejszają ryzyko wejścia przez znane podatności. Dotyczy to szczególnie usług wystawionych do Internetu (zdalny pulpit, VPN, serwery WWW, poczta).
      Uwierzytelnianie wieloskładnikowe - MFA znacząco utrudnia przejęcie kont, zwłaszcza dostępu zdalnego i kont uprzywilejowanych (administratorzy).
      Zasada najmniejszych uprawnień - użytkownicy i usługi nie powinni mieć większych uprawnień niż to konieczne. To ogranicza skalę szkód, gdy ktoś przejmie konto lub komputer użytkownika.
      Segmentacja sieci - podział sieci na strefy i ograniczenie dostępu pomiędzy nimi utrudnia „rozlanie się" ataku na serwery i zasoby krytyczne.
      Ochrona i monitorowanie stacji - programy ochronne i monitorujące (antywirus/EDR) potrafią wykrywać typowe zachowania ransomware (np. masowe modyfikacje plików), ale nie zastępują kopii zapasowych i dobrych uprawnień.
      Procedury reagowania - warto z góry wiedzieć, co robić po wykryciu incydentu: odłączenie zainfekowanej maszyny od sieci, zabezpieczenie logów, powiadomienie administratora, oraz odtwarzanie z kopii.
- Podsłuch połączenia sieciowego (ang. *sniffing*) jest popularnym rodzajem ataku, który często prowadzi do utraty tajności danych. W zależności od okoliczności podsłuch lub uzyskanie informacji w ten sposób może stanowić czyn zabroniony (patrz Kodeks karny (ISAP)). W wyniku podsłuchu można uzyskać między innymi: hasła, identyfikatory kont oraz dane prywatne. Podsłuch jest atakiem biernym (nie czyni więc bezpośrednich szkód w systemie), ale bardzo niebezpiecznym, ponieważ najczęściej stanowi wstęp przed zastosowaniem innych metod. Najważniejszym narzędziem w walce z podsłuchem połączenia sieciowego w przypadku połączeń kablowych jest segmentacja sieci i ograniczanie dostępu do ruchu, który nie jest przeznaczony dla danego hosta. Przy transmisji danych należy stosować bezpieczne protokoły oraz szyfrowanie. Należy także unikać przesyłania haseł w formie jawnej. W przypadku sieci bezprzewodowej bardzo ważne jest użycie współczesnych, bezpiecznych standardów Wi-Fi (np. WPA2-AES lub WPA3) oraz unikanie przestarzałych rozwiązań (np. WEP). Niezależnie od tego, w sieciach publicznych i nie w pełni zaufanych warto polegać na szyfrowaniu warstwy aplikacji (np. https) albo na tunelowaniu ruchu (VPN). W przypadku sieci komórkowych pamiętajcie, że starsze technologie (np. 2G) mają znane słabości, a część ataków polega na wymuszeniu przełączenia telefonu na słabszy standard; dlatego i tu sensowne jest szyfrowanie end-to-end i bezpieczne protokoły.
- Łamanie haseł. Najczęściej stosowanymi metodami ataku w celu uzyskania nieautoryzowanego dostępu są tzw. metody korzystające z haseł. Polegają one na podjęciu próby przeniknięcia do systemu poprzez podanie identyfikatora użytkownika i hasła. Atakujący może próbować wielu haseł, aż do momentu podania właściwego. Szybko zorientowano się, że nie jest trudno napisać program generujący rozmaite hasła. Obecnie istnieje wiele tego typu programów, które skutecznie działają na różnych platformach systemowych. Używają one zbiorów słów (słowników), dlatego też ataki tego typu znane są jako metody słownikowe. Programy takie mogą również próbować wszystkich możliwych kombinacji znaków, jakie mogą wystąpić w haśle – jest to wtedy tak zwany atak brutalny lub wyczerpujący. Skutecznym sposobem obrony przed atakami na hasła jest stosowanie długich i unikalnych haseł lub passphrase, ograniczanie liczby prób logowania, stosowanie sprawdzonych mechanizmów powiadamiania o próbach nieautoryzowanego dostępu oraz bezpieczne przechowywanie danych uwierzytelniających (patrz NIST SP 800-63B).
- Podszywanie się (ang. *spoofing*) pod inną maszynę z sieci może wystąpić na każdej z warstw protokołu TCP/IP (spoofing adresu sprzętowego, spoofing ARP, spoofing routingu IP, spoofing nazw DNS, spoofing połączeń TCP). Komputery wymieniające między sobą informacje przekazują nawzajem dane o sobie. Osoba niepowołana może wysłać do komputera-odbiorcy fałszywe informacje o swoim komputerze, które świadczą, że jest on bezpiecznym komputerem głównym znajdującym się wewnątrz sieci lub poza nią. Podczas ataku tego typu pakiety wysyłane przez intruza mają dostęp do systemu, do którego się on włamuje i do usług tego systemu.
- Zablokowanie usługi – DoS (ang. *Denial of Service*) jest to przerwanie dostarczania usługi spowodowane zniszczeniem systemu lub jego chwilową niedostępnością. Zagrożenie to dotyczy konkretnych usług realizowanych przez system, takich jak WWW, FTP czy poczta elektroniczna, lub też całego serwera. Ataki DoS najczęściej są atakami zewnętrznymi, odbywającymi się z zewnątrz sieci lokalnej na przykład z Internetu. Często atakujący za pomocą techniki spoofingu ukrywa swój prawdziwy adres internetowy. Współcześnie częściej

spotyka się DDoS (ang. *Distributed Denial of Service*) – atak z wielu urządzeń (np. przejętych komputerów i innych urządzeń sieciowych, tworzących *botnet*) (patrz CISA: Understanding and Responding to DDoS Attacks oraz NIST Glossary: botnet). Ochrona przed takim atakiem jest trudna i w praktyce wymaga zarówno mechanizmów po stronie usługodawcy, jak i współpracy z dostawcą Internetu.

Wirusy komputerowe

Wirusy i konie trojańskie należą do grupy programów powodujących zakłócenie pracy systemu informatycznego. Pod pojęciem wirus komputerowy rozumie się program, który potrafi się rozmnażać i dopisywać w postaci ukrytej do innych programów lub w sektorach rozruchowych dysków. Najlepszym sposobem ochrony przed wirusami komputerowymi jest rozsądek - czyli korzystanie tylko z bezpiecznych źródeł danych i oprogramowania, w połączeniu ze stosowaniem oprogramowania ochronnego. Powszechną grupę wśród wirusów stanowią makrowirusy, czyli wirusy tworzone za pomocą języków makropoleceń dostępnych na przykład w edytorach tekstów takich jak MS Word. Zauważcie też, że infekowanie MBR było częstsze w przeszłości; współcześnie istotną rolę w ochronie rozruchu odgrywają mechanizmy takie jak UEFI i Secure Boot, a ataki częściej idą innymi drogami.

Komentarz

Zagrożenie złośliwym oprogramowaniem często dotyczy w największej skali systemów z rodziny Windows, ale malware dla systemów Unixowych i macOS również istnieje. W środowiskach firmowych spotyka się więc warstwową ochronę: aktualizacje, ograniczanie uprawnień, kontrolę uruchamiania oprogramowania, monitorowanie (EDR) oraz w razie potrzeby oprogramowanie antywirusowe.

Awarie sprzętu

Przyczyną utraty danych może być też awaria sprzętu. Wiele systemów nie może pozwolić sobie na utratę danych lub przestoje spowodowane awarią sprzętu. Aby osiągnąć odpowiednią niezawodność stosuje się następujące metody:

- wykonywanie stałych kopii zapasowych.
  Kopie zapasowe powinno wykonywać się regularnie. Jest to jedna z prostszych i mniej kosztowych metod zapewnienia bezpieczeństwa danych. Powinna być stosowana nawet przez pojedynczego użytkownika – w tym również przez Ciebie drogi Czytelniku. Jednak wykonywanie kopii bezpieczeństwa nie rozwiązuje całkowicie problemu awaryjności sprzętu, gdyż pozwala na odzyskanie danych aktualnych w momencie wykonania ostatniej kopii.
- macierze RAID
  Stanowią one zestaw kilku dysków magnetycznych traktowanych przez system operacyjny jak jeden dysk logiczny. Istnieje wiele rozwiązań tego typu różniących się szybkością działania, możliwością i szybkością odtwarzania danych oraz kosztami eksploatacji.
- zasilacze awaryjne
  przed zanikiem napięcia zasilającego można uchronić stosując specjalne akumulatorowe zasilacze awaryjne (UPS) lub generatory prądotwórcze. Urządzenia takie pozwalają na normalną pracę systemu od kilku minut do kilku godzin. Praca zasilaczy awaryjnych może być nadzorowana przez system operacyjny.

1.3. Kryptografia

Kryptografia dostarcza narzędzia dzięki którym można, stosując metody matematyczne, zabezpieczać zarówno dane gromadzone w lokalnych magazynach informacji jak i dane przesyłane przez sieć.

Szyfrowanie informacji ma szczególne znaczenie, jeśli korzystamy z systemów otwartych takich jak na przykład Internet. Rozwój bezpiecznych metod kryptograficznych w znaczący sposób przyczynił się do rozwoju handlu elektronicznego i bankowości elektronicznej.

Funkcje skrótu, czyli przekształcenia jednokierunkowe

Zanim przejdziemy do opisu pełnoprawnych szyfrów, przyjrzyjmy się działaniu z pozoru bez sensu: czy da się przekształcić informację tak, aby nikt (nawet autor przekształcenia) nie mógł jej „odwrócić" do postaci pierwotnej? Jest to możliwe dzięki stosowaniu tzw. **funkcji skrótu** (ang. *hash functions*). Zauważcie jednak ważną rzecz: funkcje skrótu nie są szyfrowaniem. Szyfrowanie zakłada możliwość odszyfrowania wiadomości (posiadając klucz), natomiast skrót jest przekształceniem jednokierunkowym.
Ogólnie rzecz ujmując - funkcja skrótu przyjmuje dowolnie długi ciąg bajtów, i przekształca go w inny ciąg nazywany **skrótem**, zazwyczaj o stałej długości, niezależnie od długości ciągu pierwotnego. Przekształcenie ponadto charakteryzuje się kilkoma cechami:

- jednostronność - praktycznie nie da się odtworzyć danych wejściowych znając tylko skrót;
- efekt lawinowy - zmiana choćby jednego bitu w łańcuchu pierwotnym powoduje kompletną zmianę wyniku przekształcenia;
- odporność na kolizje (dla zastosowań kryptograficznych) - trudno jest znaleźć dwa różne wejścia dające ten sam skrót.

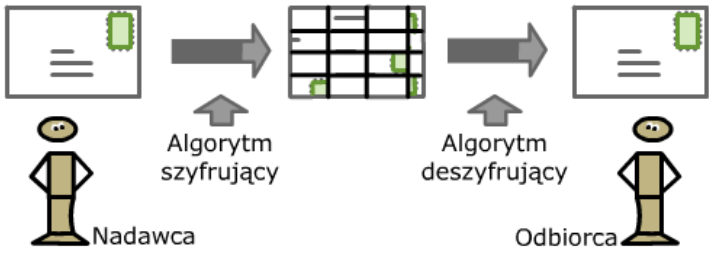
Historycznie popularnym algorytmem skrótu był **MD5**, ale dziś traktuje się go jako przestarzały i niezalecany do zastosowań bezpieczeństwa ze względu na znane problemy (patrz IETF RFC 6151). W praktyce współczesnej jako standardy spotkacie przede wszystkim rodzinę **SHA-2** (patrz NIST FIPS 180-4) oraz **SHA-3** (patrz NIST FIPS 202).
W oparciu o funkcje skrótu działa też mechanizm weryfikacji haseł w wielu systemach operacyjnych. Zauważcie - hasła nie powinny być przechowywane na dysku w postaci jawnej, tylko w postaci odpowiednio przygotowanego skrótu. Gdy wpiszemy hasło, system przekształca je i porównuje z wartością zapisaną w bazie. Jeśli zgadzają się postacie zapisane, to znaczy, że wpisaliśmy ten sam ciąg znaków. Jeśli natomiast ktoś uzyska dostęp do pliku z tymi danymi, to z samego skrótu nie powinien być w stanie łatwo odtworzyć oryginalnego hasła - jedyne co może zrobić, to próbować je „zgadnąć". W praktyce dlatego stosuje się dodatkowe mechanizmy, takie jak **sól** (losowy dodatek do hasła) oraz funkcje do haszowania haseł o wysokim koszcie obliczeniowym i/lub pamięciowym, aby utrudnić szybkie testowanie milionów kandydatów (patrz NIST SP 800-63B-4).
Sól (*salt*) to losowa wartość (zwykle ciąg bajtów), którą system dodaje do hasła przed wyliczeniem skrótu i zapisuje razem z wynikiem. Zauważcie, że sól nie jest tajna: może być przechowywana obok skrótu w bazie użytkowników. Jej rola polega na tym, że nawet jeśli dwie osoby ustawią identyczne hasło, to po dodaniu różnych soli ich zapisane wartości będą różne. Dzięki temu utrudnia się też użycie gotowych „tablic tęczowych" (prekomputowanych zestawów skrótów dla popularnych haseł), bo atakujący musiałby przygotowywać takie tablice osobno dla każdej soli.

W praktyce mechanizm wygląda tak: przy ustawianiu hasła system generuje sól, łączy ją z hasłem (np. dopisuje przed lub po), a następnie oblicza wynik funkcji służącej do haszowania haseł i zapisuje: sól + wynik. Przy logowaniu system odczytuje sól, ponownie liczy skrót z podanego hasła i tej samej soli, i porównuje z wartością zapisaną. Sama sól nie zastępuje „mocnego" haszowania: dlatego współcześnie łączy się ją z funkcjami o wysokim koszcie obliczeniowym i/lub pamięciowym (takimi, które są celowo wolne), aby dodatkowo utrudnić masowe zgadywanie haseł po wycieku bazy.

Szyfrowanie

Szyfrowanie jest metodą powszechnie stosowaną do ochrony informacji. Choć metody szyfrowania zmieniały się na przestrzeni wieków, to ogólny mechanizm wygląda ciągle tak samo. Można go przedstawić w następujący sposób:

1. Tekst jawny zostaje poddany obróbce zgodnie z algorytmem szyfrowania, czyli podlega kodowaniu. W wyniku tej operacji otrzymuje się tekst, który można co prawda czytać (w sensie odczytu znaków), ale nie można poznać jego sensu.
2. Tekst zaszyfrowany przesyłany jest do odbiorcy normalnymi, niechronionymi kanałami.
3. Po odebraniu tekstu zaszyfrowanego, odbiorca stosując algorytm deszyfracji dekoduje go do postaci czytelnej.



Proces szyfrowania i deszyfrowania

Aby można było stosować opisaną powyżej metodę do bezpiecznego przesyłania informacji, to odczytanie tekstu zaszyfrowanego bez znajomości klucza powinno być bardzo trudne lub niemożliwe. Istnieje wiele metod spełniających te wymaganie. Ogólnie wyróżniamy dwie metody szyfrowania: symetryczne i asymetryczne.

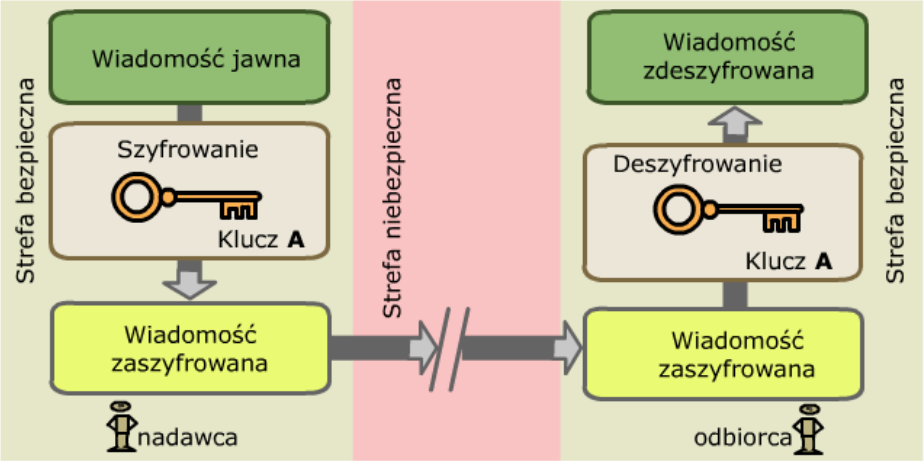
> ## Komentarz
>
> Zauważcie jeszcze jedną rzecz: w praktyce chcemy zwykle nie tylko tajności (poufności), ale także integralności i uwierzytelnienia danych - czyli wykrycia, że wiadomość nie została zmieniona po drodze, oraz potwierdzenia kto ją wysłał. Dlatego nowoczesne protokoły używają zestawu mechanizmów: szyfrowania oraz uwierzytelniania wiadomości (np. tryby AEAD, takie jak AES-GCM lub ChaCha20-Poly1305). Dobrym przykładem jest TLS 1.3, gdzie szyfrowanie jest połączone z ochroną integralności (patrz IETF RFC 8446).

Szyfrowanie symetryczne

W szyfrowaniu symetrycznym zarówno odbiorca jak i nadawca posługują się tajnym kluczem służącym do szyfrowania i deszyfrowania wiadomości (dlatego symetryczne). Przechwycenie tego klucza powoduje możliwość odczytania każdej zaszyfrowanej wiadomości.

Aby można było stosować tę metodę komunikacji muszą być dostępne następujące elementy:

- ogólny algorytm szyfrowania *E*;
- ogólny algorytm deszyfrowania *D*;
- tajny klucz (lub klucze) służący do szyfrowania i deszyfrowania informacji.



Szyfrowanie symetryczne

Szyfry symetryczne są powszechnie stosowane przez protokoły bezpieczeństwa, jako klucze sesji w poufnej komunikacji w trybie on-line. Na przykład protokół IPSec wykorzystuje symetryczne klucze sesji ze standardowymi algorytmami stosowanymi do szyfrowania i deszyfrowania poufnej komunikacji między stronami. Dla każdej poufnej sesji komunikacji używane są inne klucze.

Szyfrowanie symetryczne jest też powszechnie stosowane przez technologie zapewniające masowe szyfrowanie trwałych danych, takich jak wiadomości e-mail czy pliki typu dokument. Protokół S/MIME stosuje klucze symetryczne do szyfrowania wiadomości poufnej poczty. W systemach operacyjnych spotkacie również mechanizmy szyfrowania danych na dysku, przy czym szczegóły rozwiązania zależą od platformy.

Historycznie istotnym symetrycznym algorytmem szyfrującym, powszechnie kiedyś wykorzystywanym, jest algorytm DES. Stworzony przez IBM na podstawie szyfru Lucifer, został zmodyfikowany przez amerykańską NSA. Zaakceptowany jako amerykański standard w roku 1977.

Algorytm jest następujący:

- Wykonujemy wejściową permutację danych (IP).
- Powtarzamy 16 razy następującą operację:
  - Przestawiamy bity danych (umieszczenie tej operacji miało na celu preferowanie rozwiązań sprzętowych, w których wykonanie tego przekształcenia sprowadza się do odpowiedniego przeprowadzenia połączeń).
  - Dane, które dostaliśmy na wejściu rundy dzielimy na dwie 32-bitowe części - lewą Li i prawą Ri.
  - Rozszerzamy przez powielanie bitów prawą część do 48 bitów, uzyskując E(Ri).
  - XORujemy E(Ri) z podkluczem dla aktualnej rundy.
  - Rozbijamy otrzymany wynik na 8 fragmentów po 6 bitów.
  - Każdy z tych fragmentów jest argumentem jednej z 8 funkcji, tzw. S-BOX-ów.
  - Łączymy wyniki S-BOXów.
  - Permutujemy uzyskany wynik.
  - Jako lewą stronę wyjścia przekazujemy prawą stronę wejścia: Li + 1 = Ri.

- Jako prawą stronę wyjścia przekazujemy lewą stronę wejścia XORowaną z odpowiednią permutacją.

Deszyfrowanie polega na zastosowaniu tych samych operacji w odwrotnej kolejności (różni się od szyfrowania tylko wyborem podkluczy, który teraz odbywa się od końca).

Gwałtowny wzrost mocy obliczeniowej komputerów doprowadził do tego, że obecnie algorytm DES nie stanowi dobrego zabezpieczenia poufności danych.

Współcześnie standardem jest przede wszystkim AES (Advanced Encryption Standard) (patrz NIST FIPS 197). Spotyka się też 3DES jako rozwiązanie przejściowe w starszych systemach, ale jest ono traktowane jako algorytm schodzący i zastępowane nowszymi rozwiązaniami (patrz NIST SP 800-131Ar2).

AES został wyłoniony w ramach konkursu ogłoszonego przez NIST. Do finału konkursu zakwalifikowało się pięć algorytmów szyfrujących (Rijndael, RC6, Mars, Serpent oraz Twofish), ze szczególnym wskazaniem na algorytm Rijndael. AES jest właśnie oparty o Rijndael. Możliwe jest w nim użycie kluczy o długościach 128, 192 i 256 bitów, i operuje on na blokach danych o długości 128 bitów (oryginalna specyfikacja Rijndael dopuszczała również bloki 192 i 256 bitowe).

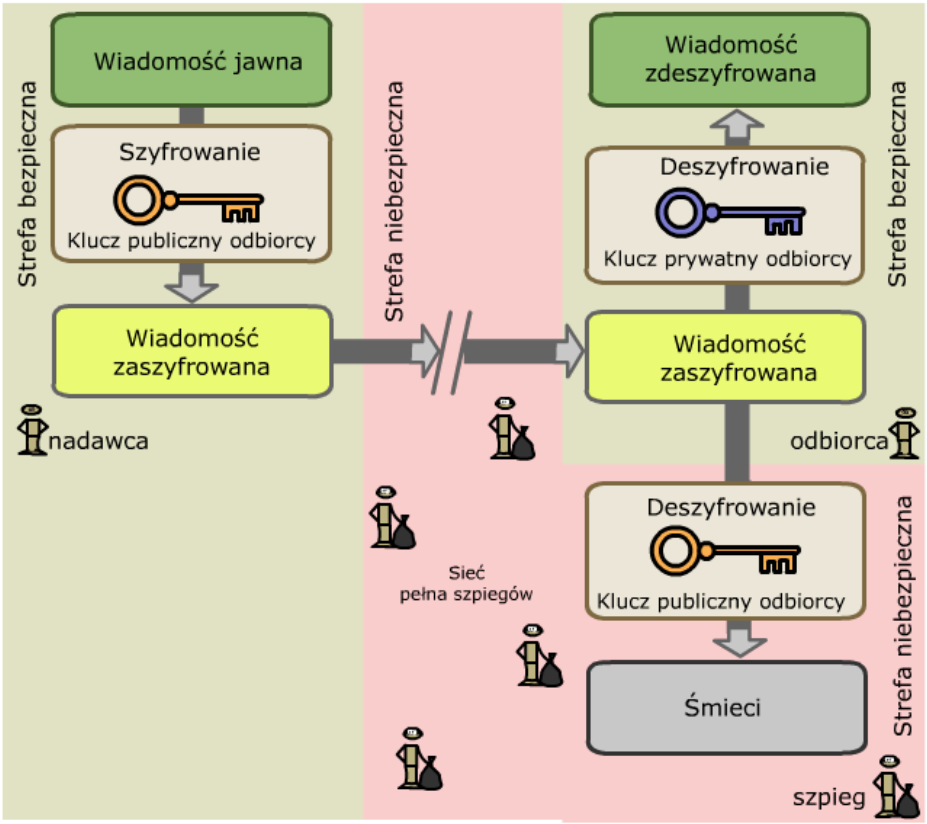
AES wykonuje rundy szyfrujące typu substitution-permutation, składające się z podstawień, przekształceń macierzowych oraz modyfikacji za pomocą klucza. Zauważcie jednak, że w praktyce o bezpieczeństwie nie decyduje tylko „sam AES”, ale też sposób użycia: tryb pracy, generowanie i przechowywanie kluczy, losowość, a także poprawna implementacja. Częściej niż „złamanie AES” spotyka się po prostu błędy wdrożenia.

Problemem w szyfrowaniu symetrycznym jest przekazywanie kluczy. Aby zapewnić bezpieczeństwo informacji klucz taki musi pozostać tajny. Nie może więc być bezpośrednio przesłany przez sieć telekomunikacyjną, gdyż byłby narażony na łatwe przechwycenie.

Problem ten rozwiązano poprzez szyfrowanie z wykorzystaniem różnych kluczy do szyfrowania i deszyfrowania. Ten sposób szyfrowania nosi nazwę szyfrowania asymetrycznego (lub publicznego). Polega on na zastosowaniu dwu kluczy: jednego do szyfrowania wiadomości, a drugiego do deszyfrowania.

Szyfrowanie asymetryczne

W szyfrowaniu asymetrycznym każdy użytkownik dysponuje dwoma kluczami. Jeden z nich to klucz jawny służący do szyfrowania wiadomości. Drugi to klucz tajny, znany tylko odbiorcy wiadomości. Służy on do odszyfrowania wiadomości.



Szyfrowanie asymetryczne

Dwaj użytkownicy mogą się skutecznie porozumiewać znając swoje klucze jawne.

Zasada działania algorytmu szyfrowania opartego na kluczach prywatnych bazuje na własnościach matematycznych dużych liczb. Polega ona na tym, że mnożenie dwu nawet dużych, specjalnie dobranych liczb jest łatwe. Trudne jest natomiast wykonanie rozkładu liczby na czynniki (o ile liczba jest odpowiednio duża i dobrze dobrana). Oznacza to, że odszyfrowanie wiadomości bez znajomości klucza tajnego wymaga

ogromnych mocy obliczeniowych (a co za tym idzie czasu) lub jest niewykonalne.

Najbardziej popularnym algorytmem z kluczem publicznym jest algorytm RSA opracowanym przez Ronalda Rivesta, Adi Shamira i Leonarda Adlemana. Bazuje on na trudności rozłożenia dużej liczby na czynniki pierwsze. Wykorzystuje funkcję Eulera Totient

$\phi(n)$

zdefiniowaną jako ilość liczb naturalnych mniejszych od *n* i względnie pierwszych z *n*. Liczby *m* i *n* są względnie pierwszymi, jeśli nie mają wspólnych podzielników innych niż 1. Funkcja $\phi(n)$ zawsze przyjmuje wartość mniejszą niż *n*. Euler odkrył, że każda liczba *k*, względnie pierwsza z *n* podniesiona do potęgi $\phi(n)$ modulo *n* daje w wyniku 1:

$$k^{\phi(n)} \bmod n = 1$$

(*a* mod *b* oznacza resztę z dzielenia liczby *a* przez *b*. Na przykład 10 mod 3 =1; 4 mod 2 = 0)

W RSA wykorzystuje się jeszcze jedną własność. Jeżeli *k* i *l* są losowymi liczbami naturalnymi będącymi odwrotnościami modulo $\phi(n)$ oraz *A* jest dowolną liczbą względnie pierwszą z *n* to:

$$(A^k)^l \bmod n = (A^l)^k \bmod n = A$$

Jeśli *A* jest częścią wiadomości wówczas szyfrowania dokonujemy za pomocą funkcji

$$S = A^k \bmod n$$

zaś deszyfrowania za pomocą funkcji

$$A = S^l \bmod n$$

Wyboru kluczy szyfrujących dokonuje się w następujący sposób:

- losuje się dwie duże liczby pierwsze *p*, *q*;
- oblicza się *n* = *pq,* oraz wyznacza funkcję $\phi(n)$
- losowo wybiera się liczbę *e* z przedziału (*1,* $\phi(n)$), względnie pierwszą z $\phi(n)$;
- wyznacza się liczbę *d* odwrotną do $e \bmod \phi(n)$, czyli $d = e^{-1} \bmod \phi(n)$
- kluczami szyfrującymi są *k* = *(n, e)* oraz *l* = *(n, d)*.

Dysponując kluczami szyfrujemy wiadomość *A*.

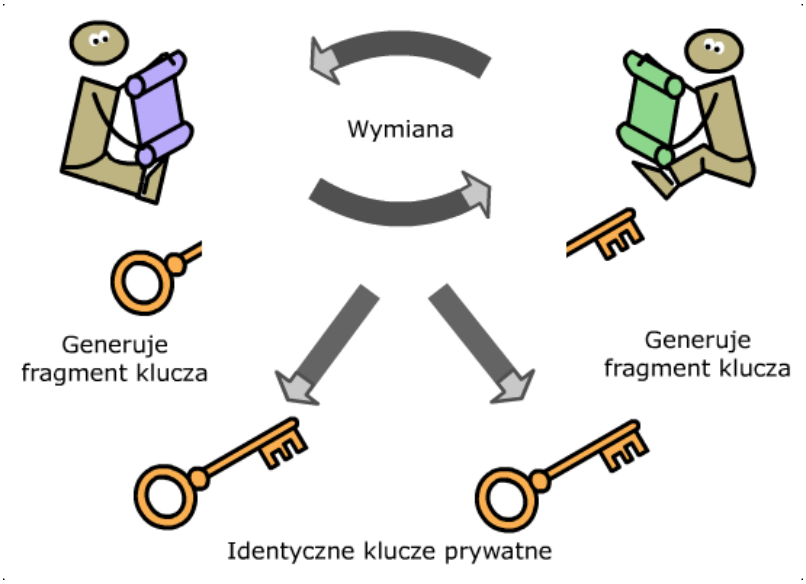
W algorytmie RSA liczby *n* oraz *k* lub *l* można wyjawiać bez poważnego narażenia bezpieczeństwa. Do złamania szyfru potrzebna jest funkcja $\phi(n)$, zaś do jej znalezienia musimy rozłożyć *n* na czynniki pierwsze, co jest zadaniem bardzo trudnym. W praktyce obecnie typowo spotyka się klucze RSA o długości co najmniej 2048 bitów, a krótsze (np. 1024 bity) traktuje się jako niewystarczające w wielu zastosowaniach (patrz NIST SP 800-131Ar2). W 1999 roku rozkład na czynniki pierwsze liczby 512-bitowej zajął trzystu szybkim stacjom około siedmiu miesięcy - dziś takie rozmiary kluczy mają znaczenie wyłącznie historyczne.

W praktyce często stosuje się połączenie metody szyfrowania symetrycznego i asymetrycznego. Ponieważ szyfrowanie i deszyfrowanie wiadomości przy użyciu szyfru symetrycznego jest dużo szybsze niż przy stosowaniu szyfrowania asymetrycznego, więc przy komunikacji dwu stron stosuje się następującą metodę. Klucze szyfru symetrycznego przekazuje się stosując metodę szyfrowania z kluczem publicznym. Klucz taki może więc być bezpiecznie przekazany przez publiczną sieć telekomunikacyjną. Następnie wiadomości mogą być już przekazywane przy zastosowaniu szyfrowania symetrycznego.

Komentarz

Współcześnie spotyka się też powszechnie kryptografię opartą o krzywe eliptyczne (np. w wymianie kluczy i podpisach), a równolegle trwają przejścia do kryptografii post-quantum. NIST zatwierdził już trzy standardy FIPS dla post-quantum kryptografii (FIPS 203, 204, 205) - jest to kierunek ważny na kolejne lata (patrz NIST CSRC: Post-Quantum Cryptography FIPS Approved).

Każdy z rozmówców generuje fragment klucza symetrycznego i przed przesłaniem do partnera szyfruje go stosując metodę szyfrowania z kluczem publicznym.



Wymiana kluczy prywatnych

1.4. Kerberos

Podstawowym, współcześnie wykorzystywanym w wielu rozproszonych systemach informatycznych, mechanizmem uwierzytelniania jest Kerberos. Zauważcie, że jest to nazwa standardu (protokołu), a nie konkretnej implementacji (patrz IETF RFC 4120: Kerberos V5).

System Kerberos został opracowany w Massachusetts Institute of Technology w latach osiemdziesiątych XX wieku. Został on zaimplementowany w wielu systemach unixowych i sieciowych, oraz w systemie Windows od wersji Windows 2000.

Kerberos jest mechanizmem uwierzytelniania z udziałem zaufanej trzeciej strony. Podkreślmy - jest to **mechanizm uwierzytelniania** a nie „sam sposób szyfrowania danych" czy „system nadawania uprawnień". Kerberos wykorzystuje kryptografię i współpracuje z mechanizmami zarządzania uprawnieniami, ale jego jądro sprowadza się do zdefiniowania kto, co i do kogo powinien wysłać, aby bezpiecznie uwierzytelnić się w sieci.

Protokół ten przyjmuje założenie, że wymiana informacji między klientami i serwerami może zachodzić w środowisku, w którym przesyłane pakiety sieciowe mogą być monitorowane i modyfikowane. Dlatego Kerberos stosuje szyfrowanie symetryczne i krótkotrwałe klucze sesji oraz bilety o ograniczonym czasie ważności. Historycznie spotykało się konfiguracje oparte na DES, ale dziś takie podejście ma znaczenie wyłącznie historyczne; w praktyce współczesnej dominuje AES, a w środowiskach Windows ważnym tematem jest odchodzenie od starszych typów szyfrowania na rzecz AES (patrz IETF RFC 4120 oraz patrz Microsoft Learn: Detect and remediate RC4 usage in Kerberos).

Kerberos zakłada, że informacje potrzebne do uwierzytelniania (dokładniej: długoterminowe sekrety/klucze principal-i) są przechowywane po stronie zaufanego serwera. Klient loguje się na maszynie klienckiej, a protokół definiuje sposób, w jaki można potwierdzić poprawność tych danych bez przesyłania hasła przez sieć w postaci jawnej. Warstwę kryptograficzną Kerberosa (m.in. sposób wyprowadzania kluczy i typy szyfrowania) opisuje standard (patrz IETF RFC 3961).

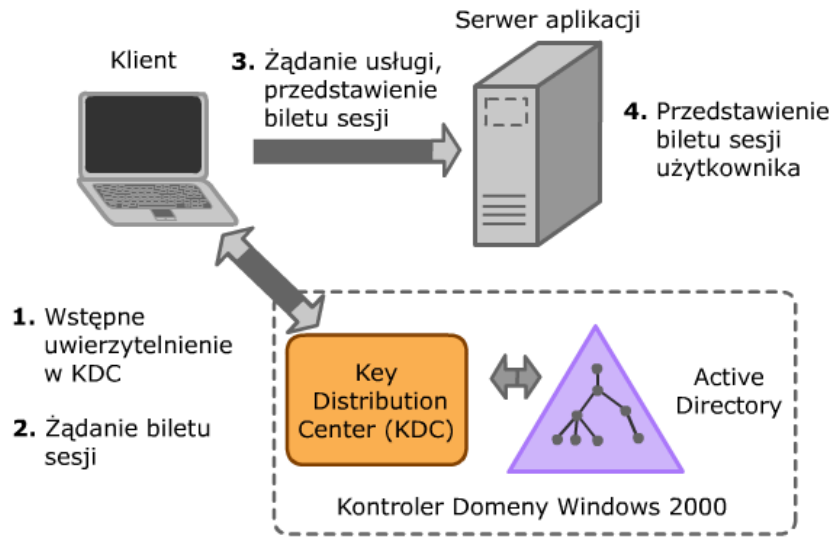
Istotnym elementem praktycznym jest czas: Kerberos używa znaczników czasu, aby utrudnić ataki typu „powtórzenie komunikatu" (replay). Z tego powodu w środowisku Kerberos zwykle wymaga się sensownej synchronizacji zegarów (np. przez NTP) oraz dopuszcza się niewielkie okno tolerancji różnic czasu.

W systemie Kerberos wyróżnić można cztery podstawowe jednostki:

- klient – stacja robocza;
- serwer uwierzytelniający AS (ang. *Authentication Server*);
- serwer przyznawania biletów TGS (ang. *Ticket Granting Server*);
- serwer aplikacji AP (ang. *Application Server*).

Procedura uwierzytelniania wymaga wymiany informacji trzech typów (tzw. transakcji). Pierwsza wymiana zachodzi między klientem i serwerem uwierzytelniającym AS. Zadaniem tej wymiany jest sprawdzenie przez serwer AS autentyczności użytkownika. Uwierzytelniony użytkownik otrzymuje informację potrzebną do zainicjowania wymiany z serwerem przyznawania biletów TGS. Druga wymiana zachodzi między klientem i serwerem TGS. W jej wyniku użytkownik otrzymuje bilet uprawniający do dostępu do określonego serwera usług AP. Trzecia wymiana zachodzi między klientem i serwerem AP. Użytkownik za pomocą biletu otrzymanego od serwera TGS może udowodnić serwerowi AP, że jest legalnym użytkownikiem.

W systemie Windows 2000 i pochodnych usługa uwierzytelniania (AS) oraz usługa przydzielania biletów (TGS) zaimplementowane są jako jedna usługa KDC (ang. *Key Distribution Center*). Usługa KDC znajduje się na każdym kontrolerze domeny (czyli komputerze, który zarządza grupą komputerów tworzących tak zwaną domenę) i korzysta z usługi Active Directory (centralna baza kontrolera domeny) jako bazy danych przechowujących konta podmiotów zabezpieczeń wraz z dodatkowymi informacjami.



Podstawowym elementem systemu Kerberos jest **bilet** (ang. *ticket*). Jest to rodzaj poświadczenia wydawanego przez KDC, które potwierdza, że użytkownik (a dokładniej: principal) został uwierzytelniony i może ustanowić sesję z określoną usługą w określonym czasie.

Kiedy komputer PC chce uzyskać dostęp do informacji przechowywanych na serwerze w obrębie tej samej domeny, musi przejść przez proces weryfikacji tożsamości w sieci. W praktyce ta weryfikacja tożsamości może zostać podzielona na sekwencję zdarzeń zachodzących pomiędzy klientem a KDC.

Kerberos stosuje komunikaty do dostarczenia każdemu z komponentów systemu Kerberos koniecznych informacji dotyczących tego co zaszło podczas procesu weryfikacji tożsamości. Jak już wspomniano, wiele z tych komunikatów jest szyfrowanych i zawiera znaczniki czasu, które utrudniają przechwycenie pakietu i jego późniejsze powtórne wykorzystanie bez wzbudzenia podejrzeń mechanizmu zabezpieczeń.

Opisywanie w pełni wewnętrznych mechanizmów Kerberos jest w ramach niniejszego podręcznika niecelowe. W ramach ciekawostki mogę Wam jedynie przedstawić, jak w uproszczeniu da się potwierdzić poprawność hasła bez wysyłania go przez sieć, ani w postaci jawnej, ani „zaszyfrowanej jakimś kluczems”. Cały proces składa się z kilku kroków:

1. Użytkownik wprowadza swój login i hasło korzystając z maszyny klienta.
2. Klient przekształca hasło w klucz (długoterminowy sekret) wykorzystywany w Kerberosie, korzystając z procedury wyprowadzania klucza (patrz IETF RFC 3961).
3. Klient przesyła do serwera uwierzytelniania (AS) żądanie rozpoczęcia uwierzytelnienia. Żądanie zawiera m.in. identyfikator użytkownika, informacje o tym jakiej usługi dotyczy uwierzytelnianie oraz dane potrzebne protokołowi, ale nie zawiera hasła w postaci jawnej.
4. AS sprawdza czy dany użytkownik istnieje, i jeśli tak - przesyła klientowi odpowiedź zaszyfrowaną przy pomocy klucza powiązanego z kontem użytkownika (sekret przechowywany w KDC). Odpowiedź zawiera m.in. klucze do bieżącej komunikacji oraz bilet TGT (*Ticket Granting Ticket*), który klient może wykorzystać do uzyskania kolejnych biletów umożliwiających korzystanie z wybranych usług.
5. Klient jest w stanie poprawnie odczytać tę odpowiedź i korzystać z TGT tylko wtedy, gdy sekret po stronie klienta (wyprowadzony z hasła wprowadzonego przez użytkownika) zgadza się z sekretem po stronie KDC. W ten sposób hasło nie musi być przesyłane przez sieć.

Po prawidłowym procesie uwierzytelniania, klient korzystając z TGT może prosić KDC (a dokładniej: część TGS) o dostęp do kolejnych usług. Jeśli uzyska bilet dla danej usługi, przekazuje go do serwera aplikacji, aby uzyskać dostęp. W podobny sposób usługa sprawdza, czy bilet jest prawidłowy, i na podstawie tego sprawdzenia udziela dostępu bądź go odmawia. Dodatkowo, bilety i klucze sesji są opatrzone ograniczonym czasem ważności, co pozwala kontrolować wygasanie uprawnień.

Opisany mechanizm jest do dnia dzisiejszego jednym z najważniejszych i najczęściej stosowanych mechanizmów uwierzytelniania w rozproszonych systemach informatycznych. Jak każdy standard, Kerberos ewoluuje - pojawiają się nowe typy szyfrowania i doprecyzowania kryptograficzne (np. AES z HMAC-SHA2; patrz IETF RFC 8009), ale ogólny przebieg procesu uwierzytelniania pozostaje podobny.

1.5. Podsumowanie

Współczesne systemy informatyczne podlegają wielu zagrożeniom zarówno wewnętrznym jak i zewnętrznym. Konieczne stało się więc zaimplementowanie mechanizmów ochronnych, pozwalających w skuteczny sposób zabezpieczyć system i gromadzone w nim informacje przed celowym lub przypadkowym uszkodzeniem, utratą bądź kradzieżą danych. Większość systemów operacyjnych dostarcza odpowiednich mechanizmów do realizacji polityki ochrony. Polityka ta jest równie ważna jak stosowane zabezpieczenia systemów. Należy więc ściśle przestrzegać ustalonych zasad.

Ja w tym miejscu dorzuciłbym proste minimum, które działa niemal zawsze: regularne aktualizacje, uwierzytelnianie wieloskładnikowe tam gdzie się da, kopie zapasowe trzymane także offline oraz zasada najmniejszych uprawnień. Do tego dochodzi edukacja użytkowników (phishing) i przećwiczone odtwarzanie działania po incydencie. To jest nudne, ale w praktyce ratuje najwięcej systemów (patrz CISA: StopRansomware Guide).

Wraz z rozwojem Internetu i usług sieciowych wzrasta również konieczność bezpiecznego przesyłania danych. Dzięki rozwojowi kryptografii i praktycznych protokołów bezpieczeństwa możemy dziś osiągać wysoki poziom poufności i integralności komunikacji w sieci publicznej. Zauważcie jednak, że kryptografia jest tylko narzędziem: o wyniku decyduje też konfiguracja, zarządzanie kluczami i certyfikatami oraz bezpieczeństwo urządzeń końcowych. Najpopularniejszymi mechanizmami wykorzystywanymi w praktyce są szyfrowanie symetryczne i asymetryczne, certyfikaty oraz podpisy elektroniczne, a w protokołach sieciowych zwykle spotkacie połączenie poufności z ochroną integralności.

W ostatnich latach coraz częściej mówi się też o kryptografii postkwantowej. W największym uproszczeniu chodzi o to, że wystarczająco silny komputer kwantowy mógłby w przyszłości znacząco osłabić część dzisiejszej kryptografii asymetrycznej (szczególnie tej opartej o faktoryzację i logarytmy dyskretne), czyli rozwiązania takie jak RSA oraz powszechnie stosowane algorytmy oparte o krzywe eliptyczne. Dla kryptografii symetrycznej skutki są zwykle łagodniejsze (w praktyce rekomenduje się po prostu odpowiednie długości kluczy), ale problem dotyczy przede wszystkim wymiany kluczy i podpisów cyfrowych.

Przejście na kryptografię postkwantową nie polega na jednym „przełączeniu opcji”, tylko na migracji całych systemów (PKI, protokołów sieciowych, urządzeń i bibliotek). NIST zatwierdził już trzy standardy FIPS dla kryptografii postkwantowej (FIPS 203, 204, 205), co jest ważnym krokiem w stronę rozwiązań „quantum-safe” (patrz NIST CSRC: Post-Quantum Cryptography FIPS Approved oraz patrz NIST: NIST Releases First 3 Finalized Post-Quantum Encryption Standards). W praktyce szczególnie ważna staje się „zwinność kryptograficzna” (możliwość wymiany algorytmów bez przebudowy całego systemu) oraz stosowanie rozwiązań przejściowych tam, gdzie kompatybilność ze starszymi urządzeniami jest nieunikniona.

Niemniej jednak nie ma (i raczej nie będzie) metody 100% zabezpieczenia systemów informatycznych. Dlatego też te naprawdę kluczowe systemy (jak np. układy sterowania instalacji przemysłowych) często chroni się poprzez ograniczanie łączności z siecią publiczną oraz budowanie stref i segmentów o różnym poziomie zaufania. Zauważcie jednak, że sama „izolacja” nie jest gwarancją bezpieczeństwa: w praktyce pojawia się zdalny serwis, aktualizacje i potrzeba wymiany danych, więc kluczowe staje się kontrolowanie zdalnego dostępu, monitoring oraz ścisłe procedury (patrz CISA: Configuring and Managing Remote Access for Industrial Control Systems). Zawsze powinniście zakładać, że incydent może nastąpić - poleganie na jednym zabezpieczeniu to zbyt wysokie ryzyko.

Spis treści

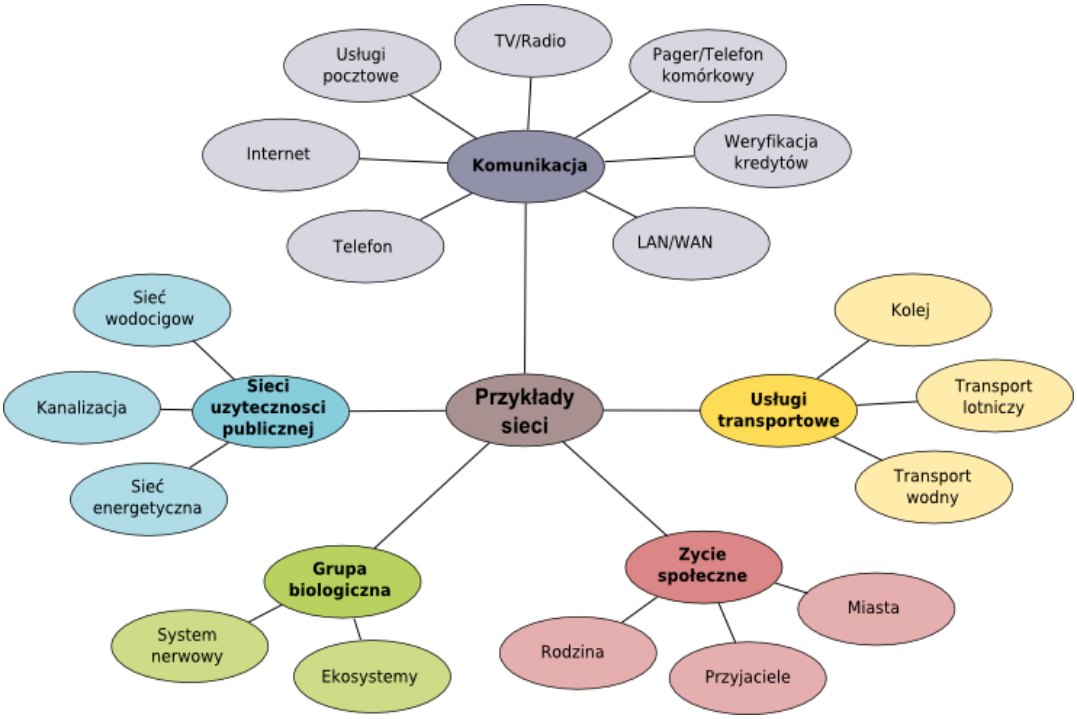
1. Sieci komputerowe
1.1. Model odniesienia OSI
1.2. Transmisja, przepustowość, media transmisyjne
1.3. Topologia sieci
1.4. Sieci LAN, MAN, WAN
1.5. Urządzenia sieciowe

1. Sieci komputerowe

Definicja

Sieć to zbiór węzłów (np. urządzeń lub ludzi) oraz połączeń pomiędzy nimi, które umożliwiają przepływ informacji, energii lub innych zasobów.

Nie zdajemy sobie często sprawy, ile sieci otacza nas w życiu codziennym. Jadąc do pracy korzystamy z sieci komunikacyjnej, gdy do kogoś dzwonimy, jesteśmy łączeni z rozmówcą za pomocą sieci telefonicznej a płacąc za zakupy kartą kredytową przesyłamy informacje za pomocą sieci teleinformatycznej. Ludzki system nerwowy oraz układ krwionośny to także sieci. Przykłady sieci można mnożyć, poniższy rysunek przedstawia kilka z nich, pogrupowanych logicznie.



Przykłady sieci występujących w naszym otoczeniu.

Sieci jako takie doczekały się też wielu prac naukowych - zajmuje się nimi zarówno matematyka, jak i nauki przyrodnicze czy społeczne. My w tym podręczniku skupimy się na sieciach komputerowych.

Ich historia zaczyna się niewiele później niż historia samych komputerów. Pierwsze komercyjnie dostępne komputery były olbrzymimi maszynami które niejednokrotnie zajmowały cały pokój / piętro. Oczywiście takie komputery nie tylko były duże, ale również drogie i trudno dostępne. W praktyce dostęp do takich maszyn miały ośrodki uniwersyteckie i (może lepiej - przede wszystkim) wojskowe. Tyle że pojawiał się problem ich optymalnego wykorzystania - nawet wojsko nie może marnować pieniędzy w nieskończoność ... ;) Ówczesne maszyny programowano albo z poziomu taśm dziurkowanych, albo prostych terminali. W obu przypadkach dostęp do nich był raczej mało efektywny – jeśli nawet każdy z użytkowników mógł „posiedzieć” przy sprzęcie jakiś fragment czasu to zawsze większą część czasu pochłaniało wklepywanie kodu programu który miał zostać uruchomiony. Powodowało to, że faktycznie większość mocy obliczeniowej tych drogich maszyn skupiała się na wykonywaniu instrukcji bezczynności. Tak więc dość szybko powstały systemy operacyjne - pozwalające na pracę wielu użytkowników na tej samej maszynie równolegle. Na maszynach pracowano korzystając z terminali, które w linii prostej wywodziły się z dalekopisów. Pierwsze terminale dostępowe do ówczesnych komputerów korzystały z różnego rodzaju łącz szeregowych, które charakteryzowały się dwoma cechami:

- w prostych, bezpośrednich łączach terminal–komputer przesyłano sygnał w postaci transmisji bazowej, co ograniczało zasięg i odporność na zakłócenia; przy większych odległościach szybko wykorzystywano urządzenia realizujące modulację (modemy) i łącza telekomunikacyjne,
- łącze szeregowe zazwyczaj służy do komunikacji typu punkt - punkt, jest to więc połączenie dwóch urządzeń, a nie sieć złożona z wielu węzłów.

W latach 60 XX wieku opracowywano teoretyczne podstawy działania sieci pakietowych (opartych na transmisji małych porcji danych pomiędzy poszczególnymi węzłami sieci), dzięki czemu uzyskiwano możliwość współdzielenia pasma przenoszenia pomiędzy wielu użytkowników z jednej strony, a z drugiej możliwość automatycznej komunikacji między dwoma urządzeniami cyfrowymi połączonymi siecią składającą się z wielu segmentów (patrz MIT OCW: Packet Switching (tło historyczne)).

W latach 60 eksperymentowano z zapewnianiem transmisji danych cyfrowych na duże odległości - wtedy też powstały pierwsze cyfrowe połączenia na odległości liczone w kilometrach. Równolegle powstawały teoretyczne opracowania dotyczące budowy rozproszonych sieci komunikacyjnych, także w wariantach cyfrowych, zdolnych na przetrwanie przewidywanej wtedy wojny atomowej. Badania te były głównie prowadzone w USA w ramach agencji ARPA. Ta sama agencja przyczyniła się do utworzenia pierwszych działających sieci pakietowych, opartych o transmisję danych w formie ramek / pakietów. W ten sposób w 1969 roku uruchomiono w Kalifornii pierwsze działające połączenia w sieci ARPANET - uważane dziś za jeden z kamieni milowych na drodze do Internetu (patrz SRI: ARPANET history).

Motywacja tych prac była dwojaka - z jednej strony chodziło o zapewnienie przetrwania infrastruktury USA w wypadku wojny atomowej, z drugiej jednak - chodziło również o ekonomiczne aspekty wykorzystania coraz popularniejszych komputerów. Zamiast wyposażać każdy ośrodek naukowy we własne komputery - taniej było połączyć ośrodki siecią i umożliwić im korzystanie nawzajem ze swoich zasobów.

Z drugiej strony - pojawienie się tanich układów elektronicznych i zbudowanych z nich komputerów osobistych pociągnęło za sobą rozwój sieci lokalnych (protoplastów dzisiejszych sieci typu Ethernet). Szybko okazało się, że niektóre urządzenia peryferyjne, takie jak chociażby drukarki, z powodzeniem mogą być współdzielone przez kilku lub nawet kilkudziesięciu użytkowników. Wspólne dane bardziej efektywnie i bezpiecznie jest przechowywać na serwerze, do którego dostęp mają wszystkie komputery, które z tych danych muszą skorzystać. Z początku komputery osobiste łączone były w małe sieci obejmujące swoim zasięgiem jeden pokój, później całe biuro a następnie budynek.

Naturalną konsekwencją rozwoju sieci było połączenie się tych dwóch podejść dzięki czemu mamy stan dzisiejszy - globalną sieć Internet, w której powoli zaczyna zacierać się różnica pomiędzy tym co lokalne a tym co dostępne zdalnie z dużej odległości. Zauważcie, że dziś ten trend jest jeszcze silniejszy: obok klasycznych sieci przewodowych powszechne są sieci Wi-Fi i komórkowe, a wiele usług i danych działa w modelu chmurowym, więc „zdalność” stała się dla użytkownika czymś naturalnym.

Lecz zanim do tego przejdziemy - pora przyjrzeć się, jak cyfrowa informacja jest przesyłana na odległość.

1.1. Model odniesienia OSI

Aspekt historyczny

Początkowy rozwój sieci komputerowych nie podlegał jakiejkolwiek standaryzacji. Różne laboratoria, firmy, ośrodki badawcze w różny sposób definiowały, czym jest sieć komputerowa, jak powinna działać, i jak ją opisywać. Doprowadziło to w krótkim czasie do kompletnego braku interoperacyjności - sieci pochodzących od różnych dostawców nie można było połączyć w jedną, spójną sieć. Dlatego też w latach 70 ubiegłego wieku Międzynarodowa Organizacja Normalizacyjna (ISO - ang. *International Organization for Standardization*) zaczęła prace nad modelem i standardami opisującymi komunikację w sieciach komputerowych. W wyniku tych prac powstała najpierw propozycja modelu (1978 rok), potem przyjęta w 1983 i opublikowana w 1984 jako model odniesienia OSI (ang. *Open System Interconnection*) (patrz ISO: ISO/IEC 7498-1). Ciekawym jest fakt, iż finalny standard tak naprawdę nie definiował i nie definiuje konkretnego rozwiązania sieci komputerowych. OSI jest tak naprawdę w dużej mierze zestawem wytycznych, czy też modelem referencyjnym do opisywania i rozumienia komunikacji w sieci komputerowej.

Model łączności, pakietowanie danych

Proces łączności sieciowej jest złożony. Dane z pierwotnej postaci cyfrowej muszą zostać przekształcone i przesłane nośnikiem fizycznym przy wykorzystaniu różnych zjawisk fizycznych (impulsów światła, fali elektromagnetycznej) do właściwego komputera docelowego, po czym muszą zostać skonwertowane z powrotem do początkowej postaci, aby stały się czytelne dla odbiorcy. Proces ten składa się z kilku etapów, uznano więc że najbardziej efektywnym sposobem jego realizacji będzie proces podzielony na warstwy. W takim procesie każda z warstw wykonuje konkretne zadanie. Ze względu na typowy sposób graficznej prezentacji tych warstw w formie stosiku leżących na sobie prostokątów, model sieci wprowadzony przez OSI czasami jest też nazywany stosem protokołów.

Sieć komputerowa służy do przesłania informacji cyfrowej - a więc bitów (zera i jedynki). Przy czym zazwyczaj tych bitów jest całkiem sporo (przykładowo - jedno zdjęcie to kilka milionów bitów), więc muszą też istnieć jakieś techniki grupowania przesyłanych bitów w większe i wygodniejsze jednostki (ramki, pakiety, itp). Podział dużej wiadomości na mniejsze jest też niezbędny by można było współdzielić jedno fizyczne łącze przez wiele wirtualnych kanałów komunikacyjnych.

Aby komputery mogły wysyłać informacje w sieci, każdy proces przesyłania rozpoczyna się w źródle, a kończy w punkcie docelowym, czyli miejscu przeznaczenia, przy czym potencjalnie istnieje wiele różnych dróg między źródłem a miejscem docelowym - jakoś więc trzeba określić tą właściwą. Przy czym musimy zadbać o sensowne wykorzystanie tych dróg. Co to oznacza? Popatrzcie na przykład: Sieci telefoniczne w latach 60 XX wieku głównie bazowały na łączach tzw. komutowanych – czyli obwodach które na czas rozmowy były zestawiane od punktu A do punktu B. Nawet jeśli rozmowa była zestawiana na dłuższe odległości np. między Nowym Jorkiem i Los Angeles – transmisja głosu wymagała połączenia obwodów między jedną lokalizacją a drugą, czyli zestawienia elektrycznej pętli między aparatami końcowymi. Jak się domyślacie - nie było to rozwiązanie w pełni efektywne, a zestawienie obwodu na czas rozmowy blokowało innym abonentom możliwość użycia tych samych zasobów transmisyjnych.

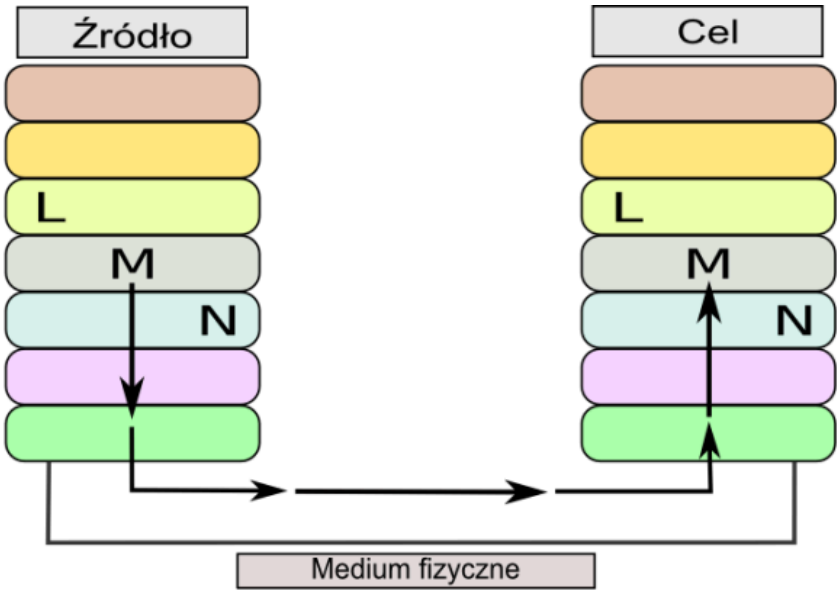
Sieci komputerowe nie korzystają z technik komutowanych - w zamian za to dzielą przesyłaną informację na małe paczuszki nazywane pakietami. Jednym z najważniejszych kroków w rozwoju sieci była transmisja danych w oparciu o przełączanie pakietów (*packet switching*) w przeciwieństwie do wcześniejszego przełączania obwodów (*circuit switching*). Sama idea przełączania pakietów i podziału transmisji na małe paczki danych była rozwijana niezależnie m.in. przez Paula Barana (RAND Corporation) oraz Donalda Daviesa (NPL), który spopularyzował termin *packet switching* (patrz Science Museum: ARPANET i packet switching). ARPANET wykorzystał te idee w praktyce, budując sieć w oparciu o dedykowane węzły pośredniczące (IMP - *Interface Message Processor*) oraz komputery końcowe korzystające z sieci. Koncepcje te były następnie dalej rozwijane i stały się podstawą sposobu myślenia o komunikacji warstwowej, w tym modelu OSI.

Informacje wędrujące w sieci nazywane są zatem danymi, pakietami lub pakietami danych. Pakiet danych to logicznie pogrupowana jednostka informacji przemieszczająca się między systemami komputerowymi. Zawiera on informacje na temat źródła i miejsca przeznaczenia, jak również inne części użytkowe (nie niosące informacji), jak przykładowo sumy kontrolne.

Protokół

Dla pakietów wędrujących ze źródła do celu w sieci ważne jest, aby wszystkie urządzenia mówiły tym samym językiem czyli np rozumiały wewnętrzną strukturę pakietu, oraz potrafiły odróżnić informację sterującą przepływem pakietów od wiadomości w nim zawartej. To właśnie definiuje protokół sieciowy - zestaw zasad, który sprawia, że łączność sieciowa jest możliwa i bardziej efektywna.

Techniczna definicja protokołu łączności brzmi następująco: zestaw zasad lub uzgodnień określający format i transmisję danych. Przy czym uzgodnienia można rozumieć na wielu różnych poziomach. Czym innym jest uzgodnienie kształtu i poziomu sygnałów elektrycznych, a czym innym określenie formatu zapytań i odpowiedzi wymienianych pomiędzy serwerem www a przeglądarką internetową. W obu przypadkach mamy do czynienia z protokołami, ale dotyczą one czegoś zupełnie innego. Dlatego też w modelu OSI podzielono je na warstwy, wychodząc z założenia, iż warstwa *n* na jednym komputerze komunikuje się z warstwą *n* na innym komputerze.



- L, M, N - warstwy w modelu komunikacji komputerowej
- M-Źródło, M-Cel - warstwy równorzędne
- ⟶ - komunikacja między warstwami równożędnymi
- Protokół warstwy M - zasady, według których M-Źródło komunikuje się z M-Celem

Warstwowy model komunikacji sieciowej.

Zadanie modelu odniesienia OSI

Model odniesienia OSI to klasyczny model komunikacji sieciowej. Pozwala on definiować funkcje sieci pełnione przez każdą z warstw. Co ważniejsze, model OSI to szkielet, którego można użyć, aby zrozumieć wędrówkę informacji w sieci. Ponadto model OSI można wykorzystać do wizualizacji procesu wędrówki informacji lub pakietów danych z programów aplikacji (np. arkuszy kalkulacyjnych, dokumentów, itd.) przez medium sieci, do innych programów umieszczonych na innym komputerze w sieci, nawet jeśli nadawca i odbiorca mają różne typy mediów sieciowych. Zauważcie też, że dziś w praktyce częściej spotkacie stos protokołów TCP/IP, ale OSI pozostaje bardzo użyteczny jako język opisu i model pomagający „poukładać w głowie" to, co faktycznie dzieje się podczas transmisji.

W modelu odniesienia OSI mamy do czynienia z siedmioma ponumerowanymi warstwami. Każda warstwa odpowiada za pewną część, aspekt komunikacji, dodatkowo definiując to, co jest wymagane na wejściu do tej warstwy, i co przekazuje ona warstwie niżej. Tak zdefiniowany podział komunikacji na warstwy ma następujące zalety:

- Dzieli komunikację sieciową na mniejsze, prostsze części, z którymi łatwiej pracować;
- Ułatwia standaryzację elementów sieciowych, co pozwala na korzystanie i obsługę z poziomu produktów wielu producentów;
- Umożliwia nawiązanie komunikacji pomiędzy różnymi rodzajami sprzętu i oprogramowania sieciowego;
- Zapobiega wpływom zmian dokonanych w jednej warstwie na inne warstwy.

Warstwy modelu OSI

Proces przemieszczania się informacji między komputerami jest podzielony na siedem mniejszych, łatwiejszych w zarządzaniu etapów. Każda z siedmiu części reprezentowana jest za pomocą własnej warstwy w modelu. Oto siedem warstw modelu odniesienia OSI:

- Warstwa 7: Warstwa aplikacji (ang. *Application layer*);
- Warstwa 6: Warstwa prezentacji (ang. *Presentation layer*);
- Warstwa 5: Warstwa sesji (ang. *Session layer*);
- Warstwa 4: Warstwa transportu (ang. *Transport layer*);
- Warstwa 3: Warstwa sieci (ang. *Network layer*);
- Warstwa 2: Warstwa łącza danych (ang. *Data link layer*);
- Warstwa 1: Warstwa fizyczna (ang. *Physical layer*).

Trzy wyższe warstwy modelu (a więc warstwy 7, 6 i 5) nazywane są warstwami aplikacji. Cztery niższe warstwy modelu (a więc warstwy 4, 3, 2 i 1) definiują sposób przesyłania danych fizycznym medium przez urządzenia sieci do końcowych stacji, i ostatecznie do warstw aplikacji.

Warstwa 1: warstwa fizyczna

Warstwa fizyczna definiuje elektryczne, mechaniczne, proceduralne i funkcjonalne specyfikacje aktywowania, utrzymywania i wyłączania fizycznego łącza między końcowymi systemami. Specyfikacje warstwy fizycznej definiują takie właściwości jak poziomy napięcia, okresy zmian napięcia, współczynniki fizycznych danych, maksymalne odległości transmisyjne, fizyczne łączniki, oraz inne, podobne atrybuty. W warstwie fizycznej określa się parametry amplitudowe i czasowe przesyłanego sygnału, fizyczny kształt i rozmiar łączy, znaczenie ich poszczególnych zestyków i wartości napięć na nich występujących, sposoby nawiązywania połączenia i jego rozłączania po zakończeniu transmisji.

Innymi słowy - warstwa ta realizuje przesyłanie strumienia bitów. Dokonuje ona konwersji bitów informacji na sygnały, które będą przesyłane w kanale z uwzględnieniem maksymalizacji niezawodności przesyłu. Co jest charakterystyczne to brak jakiejkolwiek analizy logicznej sygnału – zarówno po stronie nadawcy jak i odbiorcy głównym i jedynym zadaniem warstwy fizycznej jest kodowanie / dekodowanie zestawu cech fizycznych sygnałów płynących w danym medium na ciąg zer i jedynek.

Warstwa 2: warstwa łącza danych

Warstwa łącza danych odpowiedzialna jest za transport danych na fizycznym łączu. Zajmuje się fizycznym (niskopoziomowym) adresowaniem, niskopoziomową topologią sieci, dostępem do mediów sieciowych oraz opcjonalnie wykrywaniem i korekcją błędów.

A bardziej po ludzku: głównym zadaniem tej warstwy jest pakowanie – rozpakowanie danych – tworzy ramki.

Warstwa łącza danych odpowiedzialna jest za przygotowanie i zaadresowanie ramek u nadawcy, u odbiorcy zaś za odbiór i konwersję strumienia bitów pochodzących z urządzeń transmisyjnych w taki sposób, aby nie zawierały one błędów. Warstwa ta postrzega dane jako grupy bitów zwane ramkami. Ramka – oprócz danych zawiera informacje umożliwiające pomyślne przesłanie danych po łączach fizycznych. To tu są tworzone i rozpoznawane granice ramki. Ramka składa się z paczki przesyłanych danych (bitów) z dołączonymi do jej początku i końca grupy specjalnych bitów (znaczników). Zazwyczaj na początku dołączane są informacje sterujące ruchem (np. adres nadawcy i odbiorcy), na końcu zaś informacje służące do weryfikacji poprawności przesłania danych (np. suma kontrolna). W zależności od technologii sieciowej warstwa łącza może oferować samo wykrywanie błędów (np. CRC), albo także retransmisję i potwierdzanie ramek (ARQ) - nie jest to regułą uniwersalną.

Te dwie pierwsze warstwy umożliwiają już działanie prostych sieci lokalnych. Natomiast w przypadku bardziej skomplikowanej struktury sieci, pozostaje nam do rozwiązania jeszcze kilka problemów.

Warstwa 3: warstwa sieci

Warstwa sieciowa steruje działaniem podsieci transportowej. Jej podstawowe zadania to przesyłanie danych pomiędzy węzłami sieci wraz z wyznaczaniem trasy przesyłu. Zwyczajowo blok danych przesyłanych na poziomie tej warstwy nazywa się pakietem. Mechanizmy tej warstwy odpowiadają za podział / łączenie pakietów i przekształcanie ich w ramki na czas ich przesyłania a następnie stosowne odtworzenie.

Skoro mówimy o przesyłaniu w sieci globalnej - potrzebne nam są globalne adresy. I takie też wprowadza ta warstwa

Innymi słowy - to ona odpowiada za to, by w globalnej i ciągle zmieniającej się sieci można było w sposób automatyczny wyznaczyć trasę (optymalną trasę) pomiędzy dowolnymi dwoma komunikującymi się hostami, korzystając po drodze z potencjalnie dużej liczby węzłów pośrednich.

W propozycji OSI na tym poziomie nie ma korekcji błędów. Zakłada się, że pakiety mogą ginąć po drodze. Przykładowym i współcześnie najważniejszym protokołem tej warstwy jest IP (w wariantach IPv4 i IPv6). Historycznie istniały też inne rozwiązania, jak IPX czy AppleTalk.

Warstwa 4: warstwa transportu

Warstwa transportu odpowiada za integralność transmisji. Jej zadaniem jest podział potencjalnie dużych paczek danych (wiadomości, strumienia danych) na pakiety, przekazanie ich do warstwy sieci, a następnie, po stronie odbiorcy - na złożeniu z otrzymanych pakietów pierwotnej wiadomości. To zadanie musi być zrealizowane przy założeniu, że niektóre pakiety mogą zaginąć po drodze, oraz że mogą podróżować różnymi drogami, vide - mogą dotrzeć w innej kolejności niż były wysłane. Ponadto mechanizmy wbudowane w warstwę transportową pozwalają rozdzielać logicznie kanały łączności pomiędzy kilka połączeń sieciowych, co w praktyce realizuje się m.in. przez numerację portów. Warto też pamiętać, że transport może być realizowany w trybie połączeniowym i niezawodnym (np. TCP) albo bezpołączeniowym (np. UDP) - wybór zależy od potrzeb aplikacji.

Z tej warstwy jako pierwszej mogą korzystać aplikacje. Pozostałe warstwy są zazwyczaj współcześnie łączone w jedną warstwę, lecz dla porządku wymienimy je tutaj:

Warstwa 5: warstwa sesji

Warstwa sesji odpowiada za zarządzanie przebiegiem komunikacji pomiędzy aplikacjami: inicjowanie, utrzymywanie i kończenie dialogu, a czasem także za mechanizmy synchronizacji. W praktyce wiele funkcji kojarzonych z warstwą sesji jest realizowanych przez protokoły i biblioteki warstwy aplikacji albo przez warstwę transportu, dlatego w klasycznym Internecie warstwa ta bywa traktowana jako „wchłonięta” przez inne elementy stosu.

W praktyce rzadko wykorzystywana. Większość protokołów łączy funkcje z tej warstwy z warstwą transportową lub aplikacyjną.

Warstwa 6: warstwa prezentacji

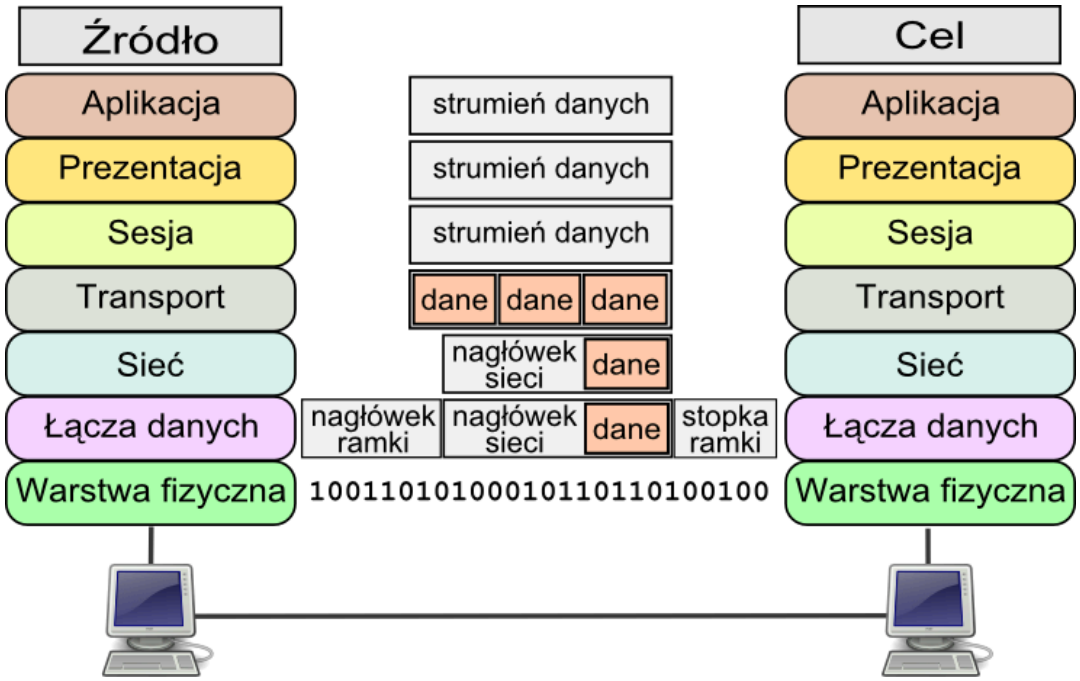
Warstwa prezentacji sprawia, że informacje wysyłane przez warstwę aplikacji jednego systemu są czytelne dla warstwy aplikacji innego systemu. Jeśli to konieczne, warstwa prezentacji dokonuje tłumaczenia wielu formatów danych wykorzystując wspólny format. Warstwa ta może być również odpowiedzialna za kompresję i szyfrowanie. Zauważcie jednak, że w praktyce szyfrowanie komunikacji w Internecie jest często realizowane przez protokoły i biblioteki „pomiędzy” transportem a aplikacją (np. TLS), co dobrze pokazuje, że OSI jest modelem myślenia, a nie sztywnym schematem implementacyjnym.

Warstwa 7: warstwa aplikacji

Warstwa aplikacji to najbliższa użytkownikowi warstwa modelu OSI. Dostarcza usługi sieciowe, na przykład dostęp do plików lub drukowanie, aplikacjom użytkownika. Tutaj znajdują się wszystkie protokoły finalne, które są bezpośrednio obsługiwane przez aplikacje serwerowe – klienckie. Do tej warstwy zaliczymy takie protokoły jak http, ftp, smb, ssh, itd ...

Enkapsulacja

Cała komunikacja w sieci zaczyna się w źródle, a kończy w miejscu przeznaczenia. Informacje wysyłane w sieci nazywane są danymi lub pakietami danych. Jeśli jeden komputer (Host A) chce wysłać dane do innego komputera (Host B), dane najpierw muszą być spakowane w procesie nazywanym enkapsulacją. Dlatego w miarę przemieszczania się danych między kolejnymi warstwami modelu OSI, zbiera on nagłówki, stopki i inne informacje. Aby zobaczyć jak działa enkapsulacja, zapoznajmy się ze sposobem, w jaki dane wędrują przez kolejne warstwy modelu OSI, co widać na poniższej ilustracji.



Enkapsulacja danych w modelu OSI.

Po wysłaniu danych ze źródła, wędrują one przez warstwę aplikacji do niższych warstw. Jak widać, pakowanie i przepływ wymienianych danych zmienia się w następstwie usług wykonywanych przez warstwy.

De-enkapsulacja

W chwili gdy oddalone urządzenie odbiera sekwencję bitów, przekazuje je warstwie łącza danych, która manipuluje ramkami. Po odebraniu ramki przez warstwę łącza danych, wykonuje ona następujące czynności:

- Odczytuje fizyczny adres i inne informacje kontrolne dostarczone przez bezpośrednio połączoną równorzędną warstwę łącza danych;
- Przeprowadza rozbiór informacji kontrolnych ramki, tym samym tworząc datagram;
- Przekazuje datagram w górę, do następnej warstwy, postępując zgodnie z instrukcjami znajdującymi się w kontrolnej części ramki;

Proces ten nazywany jest de-enkapsulacją. Każda kolejna warstwa przeprowadza podobny proces.

1.2. Transmisja, przepustowość, media transmisyjne

Transmisja analogowa i cyfrowa

Transmisja sygnału w sieciach komunikacyjnych odbywa się po łączach, nazywanych również mediami fizycznymi. Pod względem rodzaju przenoszonego sygnału, łącza można podzielić na łącza cyfrowe i analogowe.

- **Łącza analogowe** - przenoszą sygnały ciągłe, modulowane, na przykład sygnał dźwiękowy, radio, analogowa telewizja. W praktyce współczesne sieci teleinformatyczne są w ogromnej większości sieciami cyfrowymi (informacja jest kodowana jako bity), choć często bywa przenoszona w medium przy użyciu modulowanej fali nośnej (czyli „analogowo" na poziomie fizyki sygnału).
- **Łącza cyfrowe** - przenoszą sygnały dyskretne, co oznacza iż na wejściu i wyjściu łącza pojawia się strumień bitów. Przy czym w łączach cyfrowych sygnał może być przesyłany bezpośrednio (np. krótkie połączenia wewnątrz urządzeń), jak i przy wykorzystaniu modulacji. W przypadku przesyłania sygnału na dalsze odległości nawet łącza „z definicji cyfrowe" wykorzystują modulację (najczęściej fazy i amplitudy) i falę sinusoidalną jako nośnik - ze względu na odporność na zakłócenia. Sygnał w postaci fali prostokątnej ma sens głównie w transmisji na niewielkie dystanse. Niezależnie od tego, czy jest to transmisja bezpośrednia, czy modulowana, urządzenia po każdej ze stron wykonują konwersję pomiędzy reprezentacją bitową a fizycznym sygnałem w medium.

Przepustowość łącza

Z naszego punktu widzenia – jako użytkowników, najbardziej interesuje nas kilka parametrów łącza, a głównym z nich jest jego przepustowość. Przepustowość łącza (ang. *bandwidth*) określana również jako pasmo oraz bardzo często potocznie nazywana „szybkością łącza" definiuje jego możliwości. Jest to miara tego, jak wiele informacji może przepłynąć między dwoma punktami łącza w danym okresie czasu. Elementarną jednostką informacji jest bit, jednostkową jednostką czasu jest sekunda a więc przepustowość łącza zwykle określana jest w „bitach na sekundę". Ponieważ dzisiejsze łącza są miliony razy szybsze, niż 1 bit na sekundę, przyjęto poniższe oznaczenia w celu określenia przepustowości łącza.

Jednostka pasma	**Skrót**	**Równowartość**	**Przykład**
Bity na sekundę	b/s *	1 b/s = podstawowa jednostka pasma	W praktyce spotykana tylko w bardzo specyficznych zastosowaniach (np. bardzo wolne łącza telemetryczne)
Kilobity na sekundę	Kb/s	1 Kb/s = 1 000 = 10^3 b/s	Historyczne połączenia modemowe (rzędu dziesiątek kb/s)
Megabity na sekundę	Mb/s	1 Mb/s = 1 000 000 = 10^6 b/s	Łącza dostępowe i sieci lokalne starszej generacji (np. 100 Mb/s), dziś częściej 1 000 Mb/s
Gigabity na sekundę	Gb/s	1 Gb/s = 1 000 000 000 b/s = 10^9 b/s	Typowa przepustowość współczesnych sieci LAN (1 Gb/s), coraz częściej 2.5/5/10 Gb/s
Terabity na sekundę	Tb/s	1 Tb/s = 1 000 000 000 000 b/s = 10^12 b/s	Agregowane przepustowości w sieciach szkieletowych; w warunkach laboratoryjnych demonstrowano nawet 22,9 Pb/s w pojedynczym włóknie (patrz NICT: world record optical fiber transmission 22.9 Pb/s).

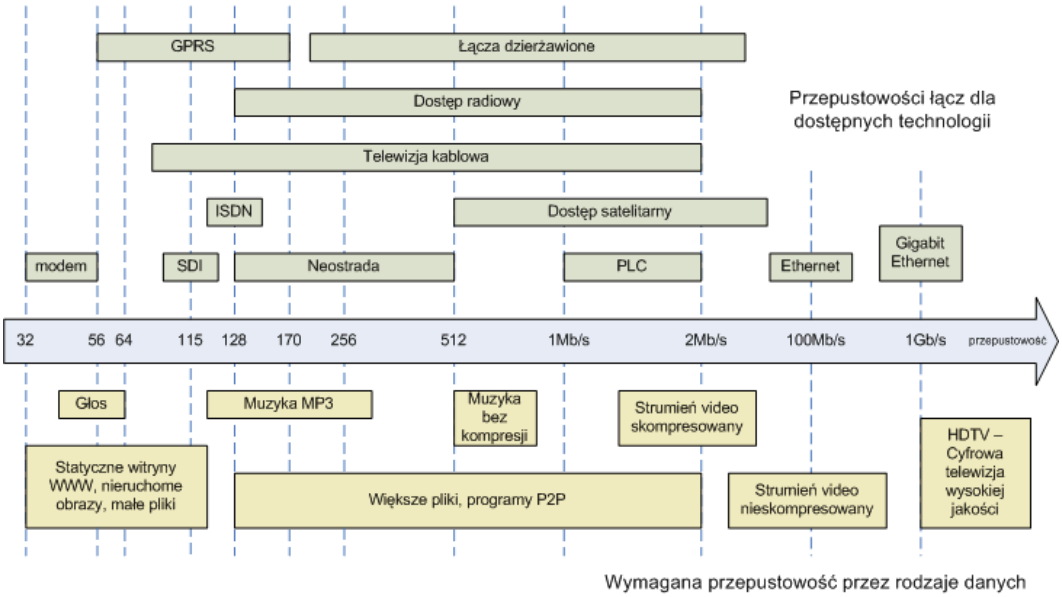
***** bity zwykle oznaczane są za pomocą małej litery „b", bajty za pomocą wielkiej litery „B". Oznaczenie b/s zastępowane jest czasem symbolem bps (ang. *bits-per-second*).

Jako użytkownicy Internetu czy innej sieci często zwracacie uwagę właśnie na przepustowość łącza, dbając o to, by była ona jak największa. Jeżeli chodzi o dostęp do Internetu to oczywiście przepustowość w dużym stopniu przekłada się na koszt opłaty abonamentowej za łącze. Podpisując umowę z dostawcą Internetu należy ustalić najodpowiedniejszy dla nas produkt biorąc pod uwagę oferowane pasmo za określoną cenę. Lecz ten parametr nie jest jedynym, który opisuje "jakość" połączenia sieciowego.

Zbliżonymi do przepustowości parametrami łącza, które powinny was interesować, to przepustowość gwarantowana, oraz cecha łącza określana jako symetryczność. Przepustowość gwarantowana określa minimalny gwarantowany w każdej chwili czasowej transfer danych. Warto na to zwrócić uwagę - zazwyczaj podawana jest jedynie przepustowość maksymalna. Maksymalna przepustowość łącza określa jedynie ograniczenia sprzętowe lub konfiguracyjne naszego dostawcy sygnału, nigdy nie mamy pewności, że taką szybkość transferu osiągniemy w danej chwili. Bardzo ważnym czynnikiem, na który nie mamy właściwie wpływu jako użytkownicy, jest ogólny ruch panujący w sieci w danej chwili czasowej (współdzielenie zasobów u operatora i po drodze w Internecie). Przepustowość gwarantowana daje nam pewnego rodzaju pewność, że globalny ruch w sieci nie wpłynie całkowicie na zablokowanie naszego łącza.

W przypadku udostępniania danych z własnych systemów komputerowych na zewnątrz istotny jest jeszcze jeden parametr łącza. Łącze symetryczne charakteryzuje się jednakową przepustowością w stronę „od" i „do" naszego komputera. Oznacza to, że z jednakową szybkością możemy odbierać i wysyłać informacje. Łącze asymetryczne charakteryzuje się różną przepustowością w przypadku pobierania danych a inną w przypadku wysyłania. Z reguły maksymalna prędkość wysyłania informacji w łączach asymetrycznych jest mniejsza niż maksymalna prędkość pobierania danych. Przykładem są łącza realizowane w sieciach telewizji kablowej (DOCSIS) lub niektóre technologie dostępowe klassy xDSL. Dla porównania: DOCSIS 3.1 jest projektowany na bardzo wysokie przepływności w dół łącza, a niższe w górę (patrz CommScope: DOCSIS 3.1 vs 4.0 (przepływności)).

Aby bardziej obrazowo przedstawić omówione zagadnienie przepustowości, poniżej znajduje się porównanie dostępnych kilka lat temu w Polsce łącz do Internetu w Polsce oraz wymagań, jakie niesie ze sobą przesyłanie pewnych określonych w swojej formie danych.



Schemat porównawczy przepustowości łącz dostępnych w Polsce i wymaganej przepustowości przez pewne formy danych.

Główną oś schematu stanowi przepustowość łącza, od najwolniejszego połączenia modemowego o prędkości 32 kb/s do zaawansowanych technologii lokalnych sieci zwanych „Gigabit Ethernet" o przepustowości 1 Gb/s. Wymagana przepustowość na dane jest w niektórych przypadkach przedstawiona orientacyjnie, to znaczy, że gdy mamy do czynienia z dźwiękiem czy strumieniem wideo, określona przepustowość na sekundę jest wymagana, aby przekaz był pełny i zrozumiały. Inaczej jest w przypadku plików, witryn WWW i pozostałych danych, które właściwie nie stanowią pewnego rodzaju transmisji „**na żywo**". W tym przypadku otworzyć witrynę czy ściągnąć plik można zawsze, liczy się natomiast czas trwania tej operacji.

Aby bardziej uzmysłowić różnice w przepustowości, policzmy czas pobierania dużego pliku z życia wziętego. Współczesna gra potrafi zajmować ponad 80 GB. Dla uproszczenia przyjmijmy 80 GB, czyli ok. 80 * 10^9 bajtów ≈ 6,4 * 10^11 bitów (1 bajt = 8 bitów). Poniżej podaję orientacyjny, teoretyczny „najkrótszy możliwy" czas pobrania (bez narzutów protokołów i bez spadków prędkości):

Rodzaj połączenia	**Przepustowość (orientacyjnie)**	**Najkrótszy możliwy czas pobrania ~80 GB**
Modem analogowy	56 Kb/s	ok. 132 dni
GPRS	do ok. 170 Kb/s	ok. 44 dni
3G	ok. 14 Mb/s	ok. 12,7 godz.
4G	w praktyce często do ok. 100 Mb/s (rzadziej wyżej); teoretycznie (LTE-A) do ok. 1 Gb/s	ok. 1 godz. 47 min (dla 100 Mb/s)
5G	od ok. 50 Mb/s do wartości rzędu 1 Gb/s i więcej (zależnie od warunków)	ok. 53 min (dla 200 Mb/s)
Dostawcy internetu (przewody)	1 Mb/s do 100 Mb/s	od ok. 7,4 dnia (1 Mb/s) do ok. 1 godz. 47 min (100 Mb/s)
Dostawcy internetu (światłowód)	100 Mb/s do 1 Gb/s	od ok. 1 godz. 47 min (100 Mb/s) do ok. 10,7 min (1 Gb/s)

W praktyce czasy są zwykle dłuższe (narzuty protokołów, ograniczenia serwera, zasięg, obciążenie sieci, limity operatora itp.), ale takie obliczenie dobrze pokazuje skalę różnic między technologiami.

Czas opóźnienia

Jednak przepustowość w przypadku sieci to nie wszystko - jest jeszcze jeden istotny parametr sieci komputerowych, szczególnie w przypadku sieci wykorzystywanych do sterowania urządzeń, czy też w grach komputerowych. Tym parametrem jest czas opóźnienia - definiowany albo jako czas podróży danych w jedną stronę, albo jako czas podróży danych tam i z powrotem.

Jego znaczenie może Wam uzmysłowić następujący przykład: Jak myślicie, jakie technicznie dostępne „łącze" zapewnia nam największą przepustowość w komunikacji transatlantyckiej? Wbrew pozorom, bywa to nie łącze satelitarne ani światłowody, tylko ogromny kontenerowiec wyładowany nośnikami danych. Płynie co prawda kilka dni, ale za to jest w stanie „przenieść" olbrzymie ilości informacji. W tym przykładzie przepustowość jest ogromna, ale opóźnienie także ogromne.

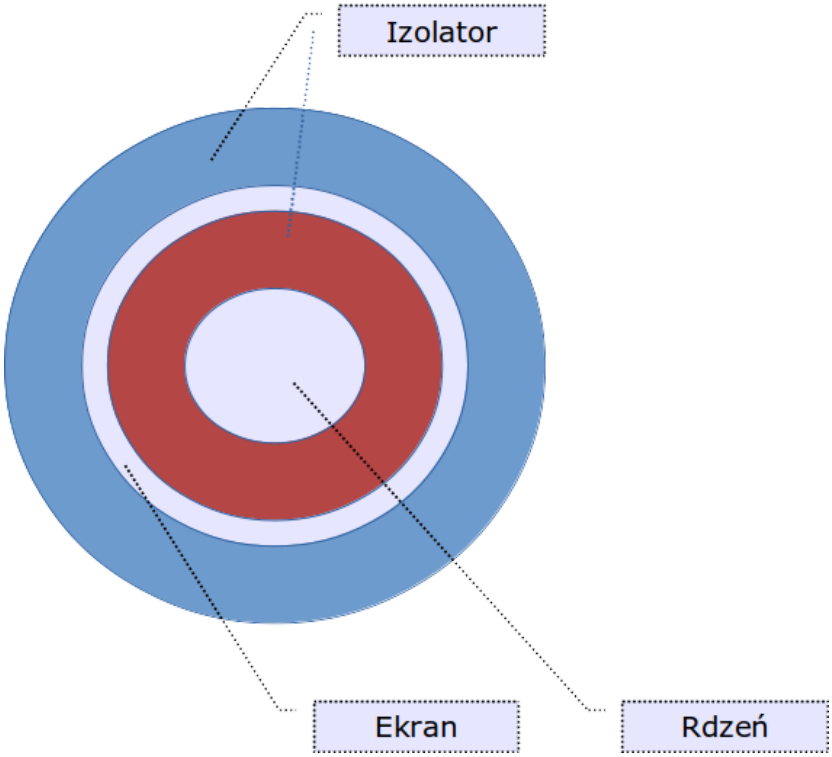
W praktyce ten parametr zazwyczaj jest podawany jako czas przesłania krótkiego komunikatu do serwera i z powrotem, i często określa się go jako PING (od nazwy programu do diagnostyki sieci komputerowych).

Nośniki danych

Nośniki danych są częścią najniższej warstwy używanej do transmisji danych w modelu OSI - warstwy fizycznej. Wszelkie dane transmitowane w sieci są reprezentowane albo jako impulsy / fala elektromagnetyczna (zmiany napięcia w czasie) na miedzianych przewodach, lub na światłowodach za pomocą impulsów świetlnych. Nośnikiem danych jest także ... brak nośnika. W przypadku sieci bezprzewodowych fala elektromagnetyczna swobodnie rozchodzi się w otoczeniu.

Kable miedziane wykorzystywane w sieciach komputerowych ogólnie dzielimy na dwa typy - kabel koncentryczny oraz skrętkę.

Kabel koncentryczny składa się z zewnętrznej, cylindrycznej osłony przewodzącej zwanej masą, która otacza pojedynczy, wewnętrzny przewód. Te dwa przewody oddzielone są warstwa izolacji tak samo jak cały kabel. W centrum kabla znajduje się przewód miedziany, natomiast zewnętrzna odizolowana warstwa to zwykle siatka druciana lub metalowa folia, działająca jako drugi przewód w obwodzie.



Budowa wewnętrzna kabla koncentrycznego.

Zewnętrzna warstwa działa jako ekran pomagający zredukować zakłócenia zewnętrzne - tworzy klatkę Faraday-a.

Ten typ kabli był bardzo popularny historycznie w sieciach lokalnych (np. Ethernet 10BASE2 i 10BASE5), gdzie wykorzystywano kabel koncentryczny 50-omowy. Typowe ograniczenia długości segmentu dla tych technologii wynosiły odpowiednio 185 m (10BASE2) oraz 500 m (10BASE5) (patrz IEEE 802.3: omówienie 10BASE2/10BASE5 (długości segmentów)). Wadą koncentryka w tego typu sieciach była m.in. mała odporność całej magistrali na pojedyncze problemy instalacyjne (np. rozłączenie lub źle zakończony odcinek) oraz kłopotliwy montaż. Z czasem więc w sieciach LAN zaczęto odchodzić od tego rodzaju kabla na rzecz skrętki.

Powrót do powszechności przyniosły kablom koncentrycznym technologie wykorzystujące okablowanie telewizji kablowej do transmisji danych (sieci HFC, standard DOCSIS). W tym przypadku transmisja danych cyfrowych odbywa się na tym samym kablu, przez który dociera do nas sygnał telewizyjny, dzięki rozdzieleniu pasm częstotliwości i zastosowaniu modulacji (patrz VIAVI: podstawy DOCSIS 3.1).

Kable koncentryczne różnią się grubością centralnego przewodu miedzianego oraz dokładnością wykonania masy. Parametry te przekładają się na tłumienie sygnału i podatność na zakłócenia, a więc pośrednio na maksymalne odległości i pasmo, które da się praktycznie uzyskać.



Dwa rodzaje kabla koncentrycznego. Końcówki kabla wykorzystywane w sieciach komputerowych niewiele różnią się od końcówek antenowych.

Kabel typu skrętka (ang. *twisted-pair*) składa się z czterech par skręconych ze sobą cienkich izolowanych przewodów miedzianych. Dzięki skręceniu odpowiednich kabli w pary wykorzystuje się efekt redukowania zakłóceń elektromagnetycznych (EMI) oraz częstotliwości radiowych (RFI) wywoływanych przez przepływ impulsów elektrycznych w każdym kablu pary. Pozwala to na ograniczenie degradacji sygnału. Aby efekt ten zaistniał, kabel musi być wykonany precyzyjnie według specyfikacji, która mówi o tym, jak wiele skrętów lub splotów jest dozwolone na określonym odcinku kabla.

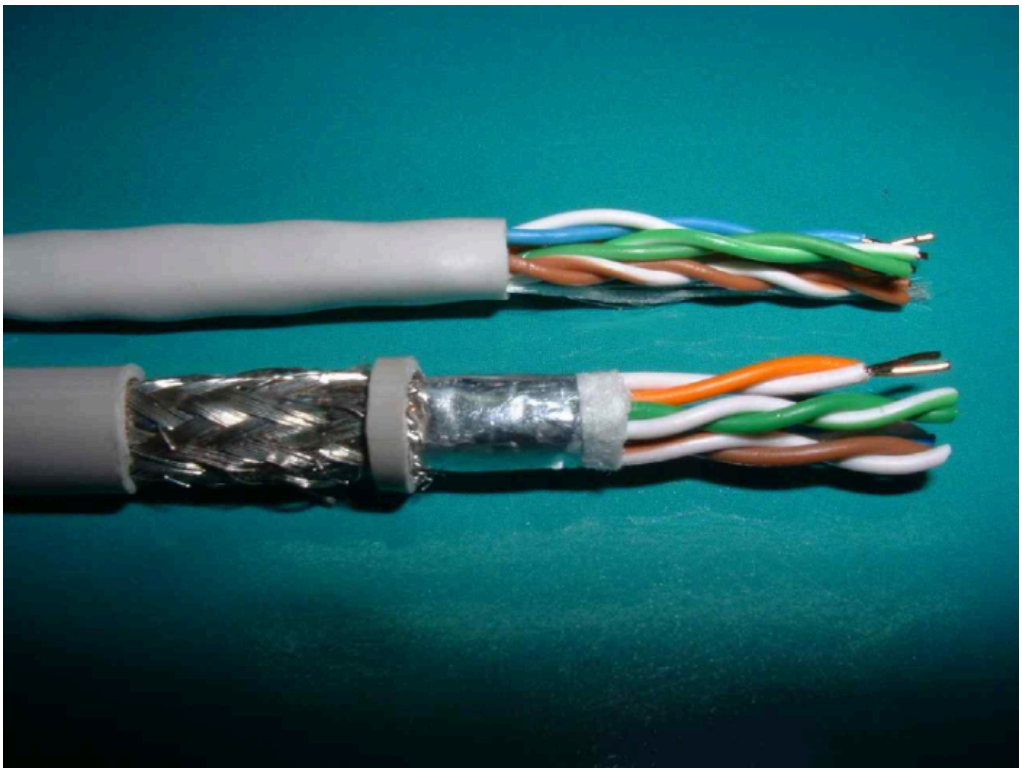
W różnych architekturach sieci i dla różnych wymagań dotyczących transmisji danych wykorzystuje się kilka odmian tego typu kabli.

Skrętka nieekranowana UTP (ang. *unshielded twisted-pair*) jest najprostszą wersją tego typu kabla. Składa się jedynie z 4 par przewodów, każdy z nich ma cienką plastikową izolację, całość otoczona jest nieco grubszą wspólną izolacją. Kabel ten ma wiele zalet, między innymi ze względu na prostotę jest tańszy od bardziej wyrafinowanych rozwiązań tego typu, z uwagi na brak dodatkowych warstw ekranujących jest cieńszy, lżejszy i bardziej giętki co ma niebagatelne znaczenie podczas instalacji. Kabel UTP może być wykorzystywany w większości architektur sieciowych i z tego powodu jest dominującym nośnikiem w sieciach LAN. Wadą skrętek UTP jest ich większa podatność na szum i zakłócenia w porównaniu do innych mediów sieciowych, a dopuszczalna maksymalna odległość między urządzeniami w typowych instalacjach Ethernet wynosi zwykle do 100 m.

Skrętka ekranowana STP (ang. *shielded twisted-pair*) łączy w sobie technikę redukowania zakłóceń oraz ekranowania. Każda para skręconych przewodów dodatkowo otoczona jest metalową folią, a wszystkie 4 pary mogą być otoczone wspólną metalową elastyczną osłoną lub folią. Skrętka typu STP redukuje szum elektryczny powstający wewnątrz kabla (przesłuch) przez zastosowanie ekranów par oraz zakłócenia poza kablem wywoływane przez inne kable i urządzenia elektryczne dzięki zastosowaniu zewnętrznego ekranu. STP może być wykorzystywana w środowiskach o wysokich zakłóceniach elektromagnetycznych, ale z powodu dodatkowych warstw ekranujących jest droższa oraz mniej elastyczna i grubsza, przez co trudniejsza w instalacji.

Pewnym kompromisem pomiędzy skrętką UTP i STP jest tak zwana **skrętka foliowana FTP** (ang. *foil twisted-pair*). W tym rodzaju kabla występuje jedynie zewnętrzny ekran wspólny dla wszystkich par przewodów w postaci metalowej elastycznej folii.

Warto też pamiętać, że dobór kategorii skrętki ma znaczenie dla maksymalnej przepustowości: na przykład okablowanie kategorii 6A jest standardowo przeznaczone do 10GBASE-T na 100 m (patrz Fluke Networks: 10GBASE-T i wymagania dla Cat 6A (100 m)).



Skrętka nieekranowana UTP (na górze) oraz ze wspólnym ekranem dla wszystkich par u dołu.



Złączka RJ45 do kabli typu skrętka.

Kabel światłowodowy to medium sieciowe, które przenosi modulowane transmisje świetlne zamiast elektrycznych jak w przypadku poprzednich nośników. Nośnik ten długo był droższy od mediów miedzianych, jednak spadek kosztów technologii optycznych sprawił, że światłowody są dziś standardem w sieciach szkieletowych, a coraz częściej także w sieciach dostępowych. Światłowód ma kilka bardzo znaczących przewag: bardzo duże możliwe przepustowości, możliwość transmisji na bardzo duże odległości oraz wysoka odporność na zakłócenia elektromagnetyczne.

W przypadku stosowania światłowodów w sieciach komputerowych, sygnały elektryczne (ciąg zer i jedynek) wysyłane z komputera zamieniane są przy wykorzystaniu diody lub lasera na impulsy świetlne, które następnie przesyłane są przez kabel światłowodowy. Z drugiej strony przewodu, impulsy świetlne zamieniane są z powrotem na sygnał elektryczny rozpoznawany przez system komputerowy lub urządzenie sieciowe.

Kabel światłowodowy używany w sieciach komputerowych składa się z włókien zamkniętych w oddzielnych osłonach. Oglądając ten kabel w przekroju można zauważyć, że warstwy ochronnego materiału zabezpieczającego, zazwyczaj plastiku oraz powłoka zewnętrzna, otaczają każde włókno optyczne. Powłoka zewnętrzna zabezpiecza cały kabel. Zazwyczaj wykonana jest z plastiku i spełnia normy przeciwpożarowe i architektoniczne.

Części przewodzące światło w światłowodzie nazywane są rdzeniem i powłoką. Rdzeń to zazwyczaj czyste szkło o wysokim współczynniku załamania. Jeśli rdzeń jest otoczony powłoką wykonaną ze szkła lub plastiku o niższym współczynniku załamania, światło nie może wydostać się poza rdzeń włókna - następuje zjawisko zupełnego odbicia światła na granicy dwóch materiałów. Proces ten nosi nazwę całkowitego odbicia wewnętrznego (patrz OpenStax University Physics: Total Internal Reflection). Dzięki niemu włókno optyczne może działać jak kanał świetlny, którym światło może wędrować na duże odległości, także pokonując zakręty.

Z uwagi na budowę światłowodu możemy wyróżnić światłowody jednomodowe i wielomodowe. Światłowód jednomodowy charakteryzuje się mniejszą średnicą rdzenia, sygnał wytworzony przez laser półprzewodnikowy ulega niewielkim zniekształceniom a fala rozchodzi się prawie równolegle do osi światłowodu. Dzięki temu światło jednego impulsu dociera do celu w prawie tym samym czasie. Światłowody te przeznaczone są do dalekosiężnej komunikacji ponieważ minimalizują zjawisko rozmycia impulsu, które wzrasta wraz z odległością.

Światłowód wielomodowy ma większą średnicę rdzenia w porównaniu do jednomodowego, w jego przypadku następuje rozbicie fali wejściowej na wiele promieni, które przez to, że odbijają się pod innym kontem od powłoki przebywają różną odległość podczas drogi do celu. Następuje zatem efekt rozmycia impulsu wraz z odległością przebytą przez światło, a tym samym ograniczenie maksymalnej odległości transmisji przy wysokich prędkościach. W przypadku światłowodów wielomodowych często stosowana jest dioda jako nadajnik (zamiast lasera).



Światłowód z założonymi końcówkami.

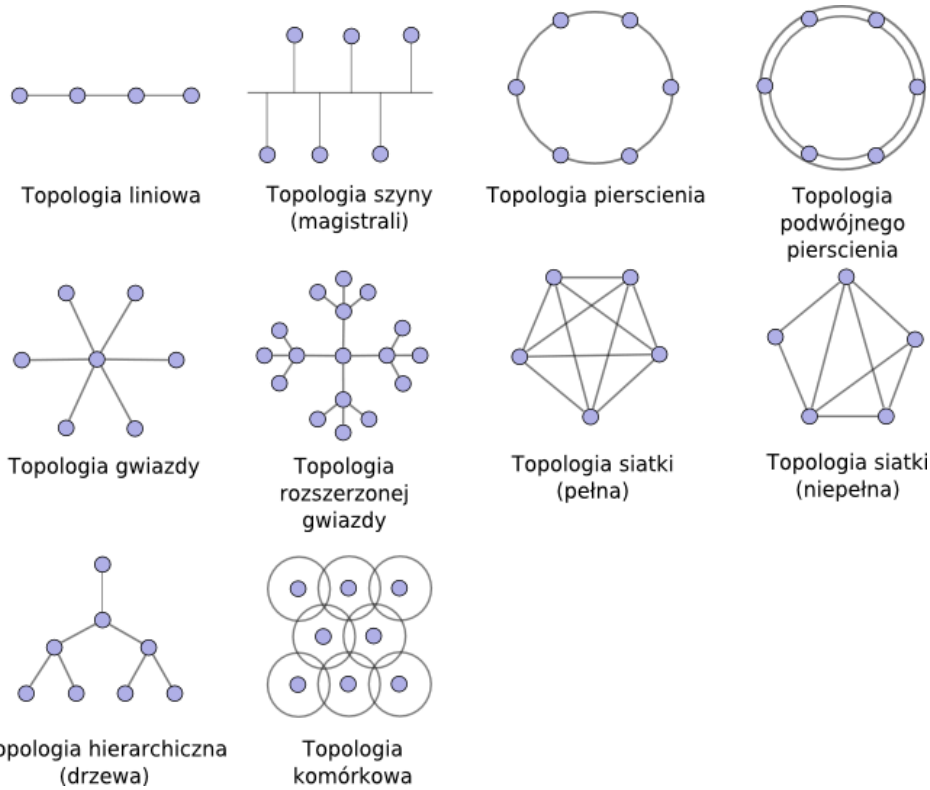
Poniższa tabela przedstawia główne cechy najpopularniejszych mediów sieciowych – maksymalną przepustowość oraz odległość. Warto pamiętać, że w praktyce zależą one od konkretnego standardu transmisji, jakości instalacji oraz użytego sprzętu.

Nośnik	**Maksymalna teoretyczna przepustowość**	**Maksymalna fizyczna odległość**
Kabel koncentryczny 50-omowy (10BASE2)	10 Mb/s	185 m
Kabel koncentryczny 50-omowy (10BASE5)	10 Mb/s	500 m
Kabel UTP kategorii 5e	1 Gb/s (typowo)	100 m
Kabel UTP kategorii 6A	10 Gb/s	100 m
Skrętka STP/FTP (w praktyce zależnie od kategorii)	1-10 Gb/s	100 m
Światłowód wielomodowy	1-10 Gb/s	od kilkuset metrów do ok. 2 km (zależnie od standardu)
Światłowód jednomodowy	1-100 Gb/s i więcej	od kilkudziesięciu do kilkuset kilometrów (z aktywnymi elementami po drodze)

1.3. Topologia sieci

Sposób łączenia urządzeń komunikujących się w sieci.

W przypadku wielu komunikujących się ze sobą urządzeń istnieje także możliwość różnorakiego ich łączenia ze sobą. Sposób łączenia urządzeń w sieci komputerowej i przesyłania między nimi informacji nazywa się **topologią** sieci. Warto od razu rozróżnić topologię **fizyczną** (jak prowadzone są kable / zasięgi) oraz **logiczną** (jak w praktyce płynie ruch i jak współdzielone są zasoby sieci). Przedstawimy Wam najczęściej stosowane topologie:



Najczęściej stosowane topologie w sieciach komputerowych. Węzły przedstawiają komputery lub urządzenia sieciowe, linie stanowią nośniki sygnału lub jego zasięg.

Topologia liniowa jest najprostszą z możliwych. Cechuje ją najmniejsze zużycie nośnika (kabla), oraz spore ograniczenia i awaryjność. W topologii tej każdy węzeł może komunikować się jedynie z najbliższymi sąsiadami, a aby skontaktować się z dalszymi jednostkami potrzebuje pośrednika. W przypadku awarii węzła lub zerwania połączenia w dowolnym punkcie sieć zostaje podzielona na 2 osobne segmenty. W praktyce tego typu topologie są stosowane jedynie w wybranych sieciach przemysłowych.

Topologia magistrali jest ulepszoną wersją topologii liniowej. Zużycie nośnika sygnału jest tu dużo większe, ponieważ każdy z węzłów musi zostać podpięty do punktu centralnego lub centralnej magistrali stanowiącej szkielet sieci. Zaletą jest to, że wszystkie węzły są ze sobą bezpośrednio połączone i mogą komunikować się ze sobą bez pośredników. W przypadku awarii węzła lub połączenia węzeł-szkielet, pozostała część sieci może nadal pracować w sposób niezakłócony. W przypadku przerwania magistrali sytuacja jest porównywalna z awarią w topologii liniowej - ze względu na konieczność występowania terminatorów na obu końcach magistrali, po jej przecięciu komunikacja staje się niemożliwa w obu "odciętych" częściach. Wadą tej topologii (w klasycznej postaci współdzielonego medium) był też problem kolizji, czyli prób jednoczesnego nadawania. Zauważcie jednak, że we współczesnych sieciach LAN opartych o przełączniki (switch) i połączenia full-duplex zjawisko kolizji na łączach przewodowych praktycznie nie występuje — dlatego magistrala w dawnym rozumieniu ma dziś głównie znaczenie historyczne. Inaczej wygląda sytuacja w sieciach bezprzewodowych: Wi-Fi współdzieli medium radiowe, więc z punktu widzenia dostępu do kanału pracuje logicznie w sposób zbliżony do magistrali (w praktyce stosuje się mechanizmy koordynacji nadawania, aby ograniczać wzajemne zakłócanie transmisji).

Topologia pierścienia tworzy jeden zamknięty pierścień składający się z węzłów i łącz. Tak naprawdę można ją przyrównać do topologii liniowej z dodatkowym przewodem „zamykającym obwód". Tak samo jak w przypadku topologii liniowej każdy węzeł komunikuje się tylko ze swoimi najbliższymi sąsiadami. W klasycznych rozwiązaniach pierścieniowych (historycznie: Token Ring) informacja krążyła zwykle w jedną stronę, a nadawanie regulował tzw. **token**. Dziś w typowych sieciach biurowych takie rozwiązania spotyka się rzadko; pierścienie wracają raczej w zastosowaniach specjalistycznych (np. przemysłowych) jako mechanizm redundancji. Zaletą jest dobre wykorzystanie teoretycznego pasma przenoszenia.

Topologia podwójnego pierścienia (historycznie kojarzona m.in. z **FDDI**) składa się z dwóch niezależnych pierścieni, które zapewniają redundancję — awaria łącza nie musi od razu rozdzielić sieci. Współcześnie podobną odporność na awarie częściej realizuje się jednak innymi mechanizmami (np. przez przełączniki i protokoły redundancji), a klasyczne FDDI należy traktować jako rozwiązanie historyczne.

Topologia gwiazdy to rozwiązanie z centralnie położonym urządzeniem aktywnym w sieci. W topologii gwiazdy węzły są promieniście połączone do urządzenia centralnego. Historycznie bywał to **koncentrator** (ang. *hub*), który rozsyłał ramki do wszystkich portów, ale we współczesnych sieciach (także SOHO) centralnym elementem jest niemal zawsze **przełącznik** (ang. *switch*), który kieruje ruch tylko do właściwego adresata. Topologia ta pozwala na wygodną komunikację wszystkich węzłów sieci – choć zawsze poprzez jednego pośrednika. Zaletą sieci jest mała awaryjność jeżeli chodzi o zrywanie poszczególnych połączeń. Uszkodzenie medium (kabla) zawsze odłącza tylko jeden węzeł. Wadą jest całkowity brak odporności na awarię węzła centralnego. Inną wadą jest większe zapotrzebowanie na kable niż w przypadku wcześniej opisywanych technologii, chociaż zależy to oczywiście od fizycznego rozlokowania węzłów sieci i punktu centralnego. Inną ciekawą cechą jest fakt, iż przepływ informacji pomiędzy wszystkimi węzłami przechodzi przez jedno urządzenie, co daje ciekawe możliwości pod względem bezpieczeństwa czy ograniczeń dostępu.

Topologia rozszerzonej gwiazdy pochodzi od topologii gwiazdy, gdzie niektóre z pośrednich węzłów topologii działają jako centra dla własnych topologii gwiazdy. Zaletą tej topologii nad jej prostszą wersją jest to, że przewody są krótsze oraz odciążony jest węzeł centralny topologii, na rzecz którego obsługę ruchu w ramach „lokalnej gwiazdy" przejmują centra pośrednie. Topologia rozszerzonej gwiazdy ma strukturę hierarchiczną i może być konfigurowana w taki sposób, aby ruch pozostawał lokalny. Taką strukturę ma obecny system telefoniczny.

Topologia siatki definiuje wiele połączeń pomiędzy różnymi węzłami w sieci, wraz z tworzeniem wielu potencjalnych dróg pomiędzy dwoma węzłami w takiej sieci. W takiej wersji (**niepełnej**), jest to typowa topologia Internetu, w założeniu pozbawiona ściśle określonej struktury. Nacisk jest jednak kładziony na to, by nigdy nie było w tej sieci tzw. **SPOF** (ang. *Single Point of Failure*), czyli by zapewnić pomiędzy dowolnymi dwoma węzłami co najmniej dwie alternatywne drogi. Jeśli zapewnimy połączenie każdego węzła z każdym mamy do czynienia z topologią **pełną** bądź zupełną. Takie graniczne podejście do połączenia węzłów w sieć ma swoje oczywiste wady i zalety, lecz bardzo rzadko wykorzystywane jest w rzeczywistych przypadkach. Sieć ta jest całkowicie odporna na awarie, zarówno węzłów jak i połączeń między nimi. Dopiero duża liczba awarii może doprowadzić do rozdzielenia sieci na oddzielne segmenty. Wadą tej topologii jest ogrom połączeń, które trzeba wykonać przy większej liczbie węzłów.

Topologia hierarchiczna, nazywana również topologią drzewa przypomina topologię rozszerzonej gwiazdy. Główną różnicą jest brak centralnego węzła, używany jest węzeł podstawowy, z którego rozchodzą się kolejne węzły. Istnieją dwa rodzaje tej topologii: drzewo binarne, gdzie każdy z węzłów ma dokładnie dwa węzły podrzędne oraz drzewo szkieletowe, gdzie węzły rozchodzą się dowolnie od pnia szkieletu. Przepływ informacji jest hierarchiczny, odporność na awarię i skala jej wpływu na sieć jest zależna od punktu awarii. Nie ma tu znaczenia, czy awarii ulegnie węzeł czy połączenie, skutki są takie same.

Topologia komórkowa jest charakterystyczna dla sieci radiowych, w których obszar pokrycia dzieli się na **komórki** obsługiwane przez **stacje bazowe**. Urządzenie w ruchu może utrzymywać połączenie dzięki mechanizmom przekazywania obsługi między stacjami (handover). W zależności od konfiguracji sieci, awaria jednego elementu infrastruktury nie powinna wpłynąć na ogólne działanie sieci, może jedynie spowodować zanik zasięgu w punkcie i okolicy uszkodzonego węzła. Nie istnieje tu problem z przerwaniem nośnika sygnału, wpływ na wydajność mogą mieć jednak czynniki zewnętrzne, np. atmosferyczne. Jest to topologia, która dopuszcza utrzymywanie połączenia urządzenia znajdującego się w ruchu, m.in. poprzez przełączanie obsługi połączenia pomiędzy poszczególnymi komórkami. Taką topologię wykorzystują współczesne sieci telefonii komórkowej.

W praktyce najczęściej spotkacie się z **gwiazdą przełączaną** (switch) oraz z sieciami bezprzewodowymi opartymi o **punkty dostępowe** (a coraz częściej także o rozwiązania typu **mesh Wi-Fi**). Jeśli spojrzymy szerzej — **Internet jako całość** jest dobrym przykładem **topologii siatki niepełnej**, w której istnieje wiele alternatywnych dróg między węzłami, ale nie ma pełnych połączeń „każdy z każdym". Magistrala i klasyczne pierścienie mają dziś głównie znaczenie historyczne lub specjalistyczne.

1.4. Sieci LAN, MAN, WAN

Podział sieci ze względu na obejmowany obszar

Sieć komputerowa może być dzielona ze względu na zasięg (obszar działania) oraz typowe zastosowania. W zależności od tych kryteriów wyróżnia się kilka klas sieci.

PAN

Sieć typu **PAN** (ang. *Personal Area Network*) to typ sieci przeznaczony do komunikacji w najbliższym otoczeniu, zazwyczaj o zasięgu rzędu kilku–kilkunastu metrów. Ogólnie zakłada się, że sieci PAN są personalizowane: dostęp do nich uzyskuje zazwyczaj jedna osoba lub niewielka grupa urządzeń należących do jednego użytkownika. Obszar działania sieci PAN można nazwać przestrzenią osobistą.

Sieci PAN mogą być realizowane zarówno przy wykorzystaniu połączeń przewodowych (np. USB), jak i bezprzewodowych (np. Bluetooth). Często cechą charakterystyczną sieci PAN jest ich tymczasowość: są tworzone doraźnie w celu realizacji prostego zadania, takiego jak wymiana plików, przesłanie kontaktu, wydrukowanie dokumentu, słuchanie muzyki lub podłączenie urządzenia peryferyjnego.

Z technicznego punktu widzenia kluczowe jest to, że sieć PAN zwykle nie jest routowana globalnie (nie działa jak „mały Internet") – łączność ma charakter lokalny, a identyfikacja urządzeń jest realizowana w ramach danego standardu (np. Bluetooth).

Zakres prędkości w sieciach PAN jest szeroki i zależy od technologii. Przykładowo Bluetooth (w praktyce: Bluetooth Classic oraz Bluetooth Low Energy) jest projektowany głównie pod komunikację krótkiego zasięgu i energooszczędność, więc realne przepływności są zwykle rzędu pojedynczych Mb/s. W Bluetooth Low Energy istnieje tryb transmisji 2M PHY, ale rzeczywista przepływność użytkowa jest niższa ze względu na narzuty protokołu (patrz Bluetooth Core Specification / omówienia Bluetooth LE). Zdarzały się też rozwiązania historyczne, takie jak Bluetooth 3.0 + HS, które wykorzystywały radio Wi-Fi do uzyskania wyższych prędkości, ale nie upowszechniły się.

Z drugiej strony mamy połączenia przewodowe, np. USB, które w nowoczesnych wersjach osiąga dziesiątki Gb/s (np. USB 3.2 do 20 Gb/s, USB4 typowo do 40 Gb/s – zależnie od implementacji i urządzeń). Takie łącza nie są „siecią" w klasycznym sensie, ale w praktyce pełnią rolę bardzo szybkiej komunikacji w przestrzeni osobistej (patrz specyfikacje USB-IF: usb.org).

LAN

Sieć **LAN** (ang. *Local Area Network*) to rodzaj sieci większej niż PAN, lecz nadal ograniczonej geograficznie i pojemnościowo. Sieć LAN zazwyczaj składa się z komputerów, nośników sieciowych, urządzeń sterujących ruchem oraz urządzeń peryferyjnych. Umożliwia ona współdzielenie danych i zasobów na ograniczonym obszarze (typowy zasięg pojedynczego segmentu to setki metrów; cała sieć LAN rzadko jest projektowana na odległości większe niż kilka kilometrów).

Technologie stosowane w sieciach lokalnych można podzielić na przewodowe (skrętka miedziana, światłowód) oraz bezprzewodowe (radio – Wi-Fi). W sieciach LAN przewodowych dominującą technologią jest Ethernet. Sieci lokalne mogą być budowane w oparciu o różne topologie, jednak najczęściej spotyka się topologię gwiazdy (w praktyce: przełączaną), a w większych instalacjach – topologię rozszerzonej gwiazdy.

Jeżeli chodzi o prędkości, to w domach i małych firmach standardem są dziś łącza 1 Gb/s, a coraz częściej również 2,5 Gb/s lub 10 Gb/s (np. dla serwerów plików/NAS i szybszych punktów dystrybucyjnych). W sieciach bezprzewodowych (Wi-Fi) przepustowość zależy silnie od standardu, pasma i warunków radiowych: wartości podawane w materiałach producentów są teoretyczne, a rzeczywiste transfery bywają istotnie niższe, szczególnie przy większej liczbie urządzeń.

MAN

Sieci **MAN** (ang. *Metropolitan Area Network*) obejmują zwykle obszar miasta lub aglomeracji (rzędu kilkunastu do kilkudziesięciu kilometrów, czasem do ok. 100 km). Historycznie były pomyślane jako miejsca koncentracji usług sieciowych w obszarach zurbanizowanych, zwłaszcza w czasach, gdy łącza do sieci rozległych miały ograniczone przepustowości. W sieciach MAN do łączenia rozproszonych sieci LAN i węzłów sieciowych najczęściej wykorzystuje się światłowody.

Współcześnie sieci MAN są zwykle budowane w oparciu o rozwiązania klasy operatorskiej, takie jak Metro Ethernet (standardy i wytyczne branżowe MEF), MPLS (IETF) oraz techniki zwielokrotniania w światłowodzie (WDM/DWDM). W sieciach dostępowych spotyka się również architektury PON, np. GPON oraz XGS-PON (ITU-T). Przykładowe odnośniki: MPLS (IETF) RFC 3031, GPON (ITU-T) G.984, XGS-PON (ITU-T) G.9807.1.

WAN

Sieć rozległa **WAN** (ang. *Wide Area Network*) jest przeciwieństwem sieci lokalnych. Jej obszar działania obejmuje setki lub tysiące kilometrów. Może ona łączyć zarówno oddziały tego samego przedsiębiorstwa, jak i węzły sieciowe należące do różnych firm. W sieciach tego typu przeważają połączenia punkt–punkt o topologii niepełnej siatki.

Najbardziej znanym przykładem sieci rozległej jest Internet, ale sieci WAN to również prywatne sieci operatorskie i korporacyjne (np. oparte o MPLS lub SD-WAN) łączące oddziały firmy w różnych miastach i krajach. Sieci rozległe charakteryzują się bardzo zróżnicowanymi prędkościami transmisji. W centrum (szkielecie) sieci możliwe jest uzyskiwanie przepustowości liczonych w terabitach na sekundę, jednak z

punktu widzenia pojedynczego hosta dostępna przepustowość bywa niższa niż w sieciach lokalnych, m.in. ze względu na współdzielenie zasobów przez bardzo wielu użytkowników.

1.5. Urządzenia sieciowe

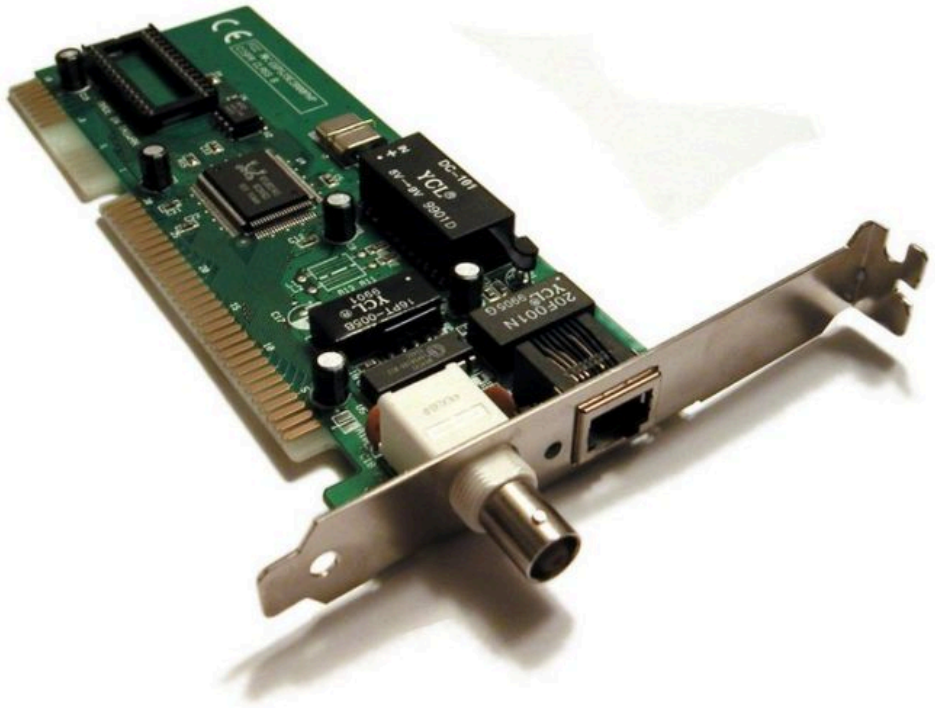
Pojęcie "host"

Urządzenia użytkownika podłączone w sposób bezpośredni do segmentu sieci często nazywane są hostami. Hosty obejmują komputery (zarówno klientów, jak i serwery), drukarki sieciowe, skanery sieciowe oraz wiele innych urządzeń. Urządzenia te mogą działać także bez podłączenia do sieci, ale ich możliwości są wtedy ograniczone.

Hosty nie są urządzeniami pośredniczącymi (takimi jak przełączniki czy routery), ale komunikują się w sieci, realizując funkcje protokołów w systemie operacyjnym i aplikacjach. W praktyce host korzysta z wielu warstw stosu (np. TCP/IP), a z medium łączy się przez interfejs sieciowy (NIC).

Karta sieciowa (NIC, ang. *Network Interface Card*) jest zwykle kojarzona z warstwą 2 modelu OSI (łącza danych), ponieważ na tym poziomie identyfikuje się w sieci lokalnej za pomocą adresu warstwy 2 (najczęściej MAC). Warto pamiętać, że interfejs sieciowy realizuje też część funkcji warstwy 1 (fizycznej), bo przetwarza sygnały transmisji danych. W zależności od medium karta sieciowa będzie miała inne złącza (np. RJ45 dla skrętki) albo antenę w przypadku Wi-Fi.

Adres MAC (ang. *Media Access Control address*) jest używany do komunikacji w warstwie łącza danych w sieciach takich jak Ethernet i Wi-Fi. Współczesne systemy i urządzenia często stosują w Wi-Fi mechanizmy poprawiające prywatność, np. losowe adresy MAC (tzw. *MAC randomization*) podczas skanowania sieci lub łączenia się z punktami dostępowymi.



Stara karta sieciowa typu combo, przeznaczona do podłączenia do kabla koncentrycznego lub „skrętki”.

Jeżeli chodzi o budowę karty sieciowej, to dawniej były to osobne płytki wkładane do gniazd rozszerzeń (np. PCI/PCIe). W praktyce dziś interfejs Ethernet i/lub Wi-Fi jest bardzo często zintegrowany z płytą główną komputera, a karty zewnętrzne spotyka się głównie w zastosowaniach specjalnych (np. dodatkowe porty, światłowód, serwery, urządzenia przemysłowe).

Typowe urządzenia sieciowe

Oprócz hostów i mediów sieciowych typowa sieć składa się także z urządzeń sieciowych łączących pozostałe elementy ze sobą. Poniżej opisujemy urządzenia, które historycznie były wykorzystywane w sieciach komputerowych, oraz te, z którymi spotkacie się dziś najczęściej. W praktyce współcześnie najważniejsze są: **przełącznik (switch)**, **router** i **access point** (lub urządzenia łączące te funkcje).

Wzmacniak (ang. repeater)

Każde medium transmisyjne ma ograniczoną maksymalną dopuszczalną długość jednego nieprzerwanego odcinka kabla. Wzmacniaki stosuje się w celu rozszerzenia sieci poza ten limit.

Wzmacniaki to urządzenia warstwy 1 (fizycznej) modelu OSI. Ich zadaniem jest zregenerowanie i „odświeżenie” sygnałów na poziomie bitu, co pozwala na transmisję na większe odległości. Wzmacniak nie analizuje ramek ani pakietów i nie potrzebuje informacji pochodzących z wyższych warstw.

W sieciach Wi-Fi podobną rolę może pełnić **range extender** lub element systemu **Wi-Fi Mesh**. Zależnie od trybu pracy urządzenie może działać jak przekaźnik/most (warstwa 2) albo jako dodatkowy punkt dostępowy, np. z uplinkiem po Ethernet. W praktyce spotkacie różne rozwiązania i nazwy marketingowe, dlatego warto patrzeć na tryb pracy oraz faktyczny sposób połączenia z siecią.

Koncentrator (ang. hub)

Koncentrator jest związany z topologią gwiazdy. Historycznie był to centralny węzeł takiej sieci. Koncentrator jest urządzeniem niskopoziomowym: nie analizuje otrzymanych ramek. Wszystkie porty koncentratora są w praktyce połączone wspólnym medium, więc gdy jedno urządzenie nadaje, wszystkie pozostałe to „słyszą", a równoczesne nadawanie powoduje kolizję.

Zadaniem koncentratora jest regeneracja i rozsyłanie sygnałów w sieci. Najważniejsze właściwości koncentratorów:

- Regenerują i „naprawiają" sygnały;
- Propagują sygnały w sieci;
- Nie filtrują ruchu sieciowego (wszystko z jednego portu jest rozsyłane na pozostałe);
- Nie wyznaczają najlepszych ścieżek.



Przykład małego (5-portowego) huba do kabli typu skrętka.

Z uwagi na wady (kolizje, wspólne medium, brak filtracji) koncentratory zostały praktycznie całkowicie wyparte przez przełączniki.

Most (ang. bridge)

Mosty to urządzenia warstwy 2 modelu OSI. Ich zadaniem jest połączenie dwóch (lub więcej) segmentów sieci LAN w taki sposób, aby ograniczyć wpływ ruchu lokalnego i kolizji jednego segmentu na drugi. Most filtruje ruch na poziomie warstwy łącza danych, dzięki czemu część komunikacji pozostaje lokalna.

Mosty filtrują ruch opierając się na adresach warstwy 2 (np. MAC). Most uczy się, które adresy znajdują się po której stronie, zapisując je w tabeli, a następnie podejmuje decyzję o przekazaniu ramki albo jej odrzuceniu.

Najważniejsze właściwości mostów:

- Są bardziej „inteligentne" niż huby: analizują ramki i przekazują je w oparciu o adresy;
- Łączą segmenty sieci LAN i ograniczają rozlewanie się ruchu;
- Tworzą więcej niż jedną domenę kolizji (w zależności od medium/technologii);
- Przechowują tablice adresów warstwy 2.

W praktyce rolę mostów przejęły przełączniki, które można traktować jak wieloportowe mosty warstwy 2.

Przełącznik (ang. switch)

Przełącznik jest urządzeniem warstwy 2 modelu OSI i w uproszczeniu można go traktować jako wieloportowy most. Przełącznik dynamicznie kieruje ramki na podstawie adresów warstwy 2 tylko do właściwego portu (adresata), zamiast rozsyłać je do wszystkich. Zmniejsza to ruch w sieci i pozwala lepiej wykorzystać przepustowość.

Z zewnątrz przełącznik może wyglądać jak hub (wiele portów), ale różnica tkwi w działaniu wewnątrz urządzenia: przełącznik „uczy się" adresów i zestawia logiczne połączenia tylko pomiędzy nadawcą i odbiorcą.

Współcześnie proste przełączniki 1 Gb/s są bardzo tanie, dlatego to podstawowe urządzenie w typowej sieci domowej i biurowej.

Router

Routery są urządzeniami warstwy 3 (sieci) modelu OSI. Pozwala to podejmować decyzje na podstawie adresów warstwy sieci (np. adresów IP), a nie adresów warstwy 2. Routery stały się podstawą Internetu opartego na protokole IP.

Zadaniem routera jest analiza pakietów warstwy 3, wyznaczanie trasy w sieci i kierowanie ruchu do właściwego interfejsu wyjściowego. Routery łączą różne podsieci, a w typowych zastosowaniach domowych realizują także funkcje takie jak NAT, DHCP i podstawowy firewall (szczegóły pojawią się w lekcji o TCP/IP).

Rozróżnić można routery sprzętowe i programowe. Routerem programowym może być komputer z co najmniej dwoma interfejsami sieciowymi, w którym system operacyjny realizuje routing między interfejsami. Routery sprzętowe to dedykowane urządzenia o różnej skali i wydajności.



Przykład niewielkiego routera.

Modem / ONT

W sieciach domowych i SOHO bardzo często spotyka się urządzenie „od strony dostawcy Internetu”, czyli modem lub ONT. **Modem** występuje w technologiach takich jak xDSL (linie telefoniczne) czy sieci kablowe (DOCSIS), a **ONT** (ang. *Optical Network Terminal*) jest zakończeniem łącza światłowodowego w lokalu klienta.

Urządzenia operatora bywają osobne albo zintegrowane z routerem Wi-Fi. W praktyce można spotkać dwa podejścia: urządzenie operatora może działać w trybie „bridge” (przekazuje łączność do naszego routera), albo może samo realizować część funkcji routingu. Wybór trybu pracy wpływa na konfigurację domowej sieci i na to, gdzie faktycznie wykonywany jest routing oraz zabezpieczenia.

Access point

Access point (AP) to urządzenie zapewniające łączność bezprzewodową Wi-Fi. Najczęściej jest podłączone do sieci przewodowej (np. Ethernet) i udostępnia sieć radiową dla urządzeń klienckich (laptopy, telefony, tablety).

Bardzo ważną funkcją AP jest kontrola dostępu oraz zabezpieczenie transmisji radiowej: ponieważ sygnał jest łatwy do odebrania przez osoby trzecie, stosuje się uwierzytelnianie i szyfrowanie (np. WPA2 lub WPA3). Poprawna konfiguracja zabezpieczeń ma kluczowe znaczenie dla bezpieczeństwa sieci domowej i firmowej.



Wygląd przykładowego access pointa.

W segmencie SOHO bardzo często spotkacie urządzenia nazywane **routerami Wi-Fi**. W praktyce łączą one w jednej obudowie kilka funkcji: router (warstwa 3), przełącznik portów LAN (warstwa 2), access point Wi-Fi, a także usługi takie jak DHCP, NAT, podstawowy firewall oraz czasem mechanizmy QoS.

Spis treści

1. Sieci TCP/IP
1.1. Rodzina protokołów TCP/IP
1.2. Adresy IP
1.3. Opis podstawowych protokołów
1.4. Protokoły i usługi stowarzyszone
1.5. Diagnostyka sieci TCP/IP

1. Sieci TCP/IP

Powoli zbliżamy się do końca naszego podręcznika. Ten rozdział skupia się na sieciach, a właściwie sieci TCP/IP - czyli podstawie działania dzisiejszego Internetu. Przedstawimy w niej stos protokołów TCP/IP, omówimy adresację IP wykorzystywaną w Internecie. Pokażemy jak tworzy się sieci i podsieci. Następnie przedstawimy protokoły i usługi aplikacji TCP/IP, oraz opiszemy podstawowe narzędzia do diagnostyki sieci.

1.1. Rodzina protokołów TCP/IP

W tej części zajmiemy się podstawami działania najważniejszej części Internetu – rodziny protokołów TCP/IP. Zadaniem tej rodziny jest przesyłanie informacji z jednego urządzenia sieciowego do drugiego w sposób możliwie niezależny od tego, jakie medium fizyczne i jaka technologia warstwy łącza jest używana po drodze. Z punktu widzenia aplikacji oznacza to jedno: program ma otrzymać spójny mechanizm komunikacji, a szczegóły „jak dokładnie bity wędrują kablami i radiem" mają zostać ukryte w niższych warstwach.

Model odniesienia TCP/IP

Omówiony wcześniej model odniesienia OSI jest uniwersalny i stanowi konstrukt teoretyczny, który może zostać wypełniony różnymi protokołami i ich kombinacjami. Założeniem była możliwość podmiany protokołów na poziomie praktycznie każdej z warstw bez wpływu na protokoły pozostałych warstw.

W praktyce, wraz z upowszechnieniem Internetu, kluczowe znaczenie zyskał stos protokołów TCP/IP (ang. *Transmission Control Protocol/Internet Protocol*), który stał się jego podstawą. Dzięki ścisłemu ustandaryzowaniu i dookreśleniu warstwy sieci (IP) oraz warstwy transportu (TCP/UDP) uzyskano możliwość komunikacji między dowolnymi dwoma komputerami znajdującymi się w dowolnym miejscu świata, niezależnie od systemu operacyjnego, na którym działają.

Protokoły TCP/IP powstały w wyniku prac badawczych prowadzonych w Stanach Zjednoczonych (m.in. w ramach DARPA) i dość szybko trafiły do systemów z rodziny UNIX (m.in. przez dystrybucje BSD), a następnie stały się standardem w łączności sieciowej. Obecnie TCP/IP jest niezaprzeczalnym fundamentem Internetu: praktycznie każdy współczesny system operacyjny ma zaimplementowaną obsługę stosu TCP/IP, a większość aplikacji sieciowych opiera się na tych samych, wspólnych zasadach przesyłania danych.

W literaturze często spotyka się określenie model warstwowy TCP/IP (czasem: model DoD lub Internet Protocol Suite) i próby porównywania go do modelu OSI. W praktyce jest to bardziej opis sposobu pracy stosu protokołów Internetu niż formalny standard-model podobny do OSI. Model TCP/IP grupuje funkcje w mniejszą liczbę warstw, a jego podział najlepiej rozumieć jako pragmatyczne uporządkowanie: co powinno dziać się w aplikacji, co w transporcie, co w routingu między sieciami i co w warstwie łącza.

Najczęściej opisuje się TCP/IP jako model czterowarstwowy:

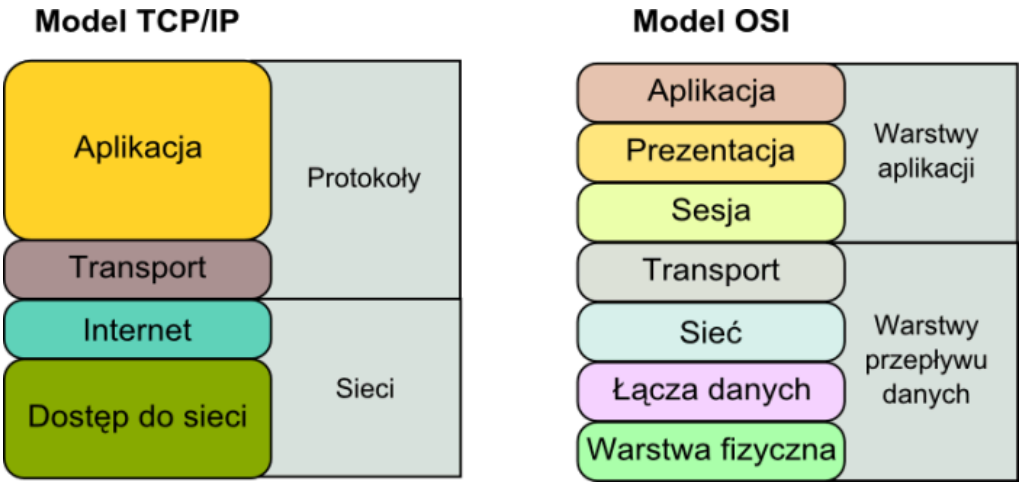
Warstwa aplikacji – obejmuje protokoły i mechanizmy, które z punktu widzenia programów „dostarczają usług sieciowych". W modelu OSI część funkcji warstwy sesji i prezentacji (np. formaty danych, negocjacje sposobu komunikacji, utrzymywanie kontekstu dialogu) nie istnieje tu jako osobne warstwy – są realizowane przez konkretne protokoły i biblioteki aplikacji. Warstwa aplikacji przygotowuje dane do przesłania i korzysta z usług warstwy transportu.

Warstwa transportu – odpowiada za komunikację typu host-do-host i zajmuje się m.in. niezawodnością, kontrolą przepływu oraz retransmisją. Najważniejszym protokołem tej warstwy jest TCP, który dzieli dane aplikacji na segmenty TCP i zapewnia niezawodny, uporządkowany strumień bajtów. TCP jest protokołem połączeniowym w sensie logicznym: utrzymuje stan połączenia (numery sekwencyjne, potwierdzenia, okna, retransmisje). Nie powstaje tu żaden fizyczny obwód jak w klasycznej telefonii, a pakiety mogą wędrować różnymi trasami. Drugim kluczowym protokołem tej warstwy jest UDP, który jest bezpołączeniowy i ma mniejszy narzut, dlatego bywa wykorzystywany tam, gdzie ważniejsza jest prostota i opóźnienia niż pełna niezawodność.

Warstwa internetowa – jej zadaniem jest przekazywanie pakietów między sieciami tak, aby mogły dotrzeć do miejsca przeznaczenia, niezależnie od tego, jakie sieci pośrednie napotkają po drodze. Protokołem tej warstwy jest IP. Routery, korzystając z tablic routingu, podejmują decyzje gdzie przekazać pakiet dalej. Można to porównać do systemu pocztowego: wysyłając list nie musimy znać wszystkich kroków pośrednich, ale zależy nam, aby dotarł do adresata.

Warstwa dostępu do sieci – nazywana też często warstwą łącza (link layer) lub host-do-sieci. Obejmuje technologie, które przenoszą pakiety IP przez konkretne medium i lokalną sieć, na przykład Ethernet, Wi-Fi, światłowody, a także rozwiązania wykorzystywane w sieciach WAN. W przybliżeniu odpowiada warstwom fizycznej i łącza danych modelu OSI.

Finalnie dochodzimy do postaci modelu jak na poniższym rysunku:

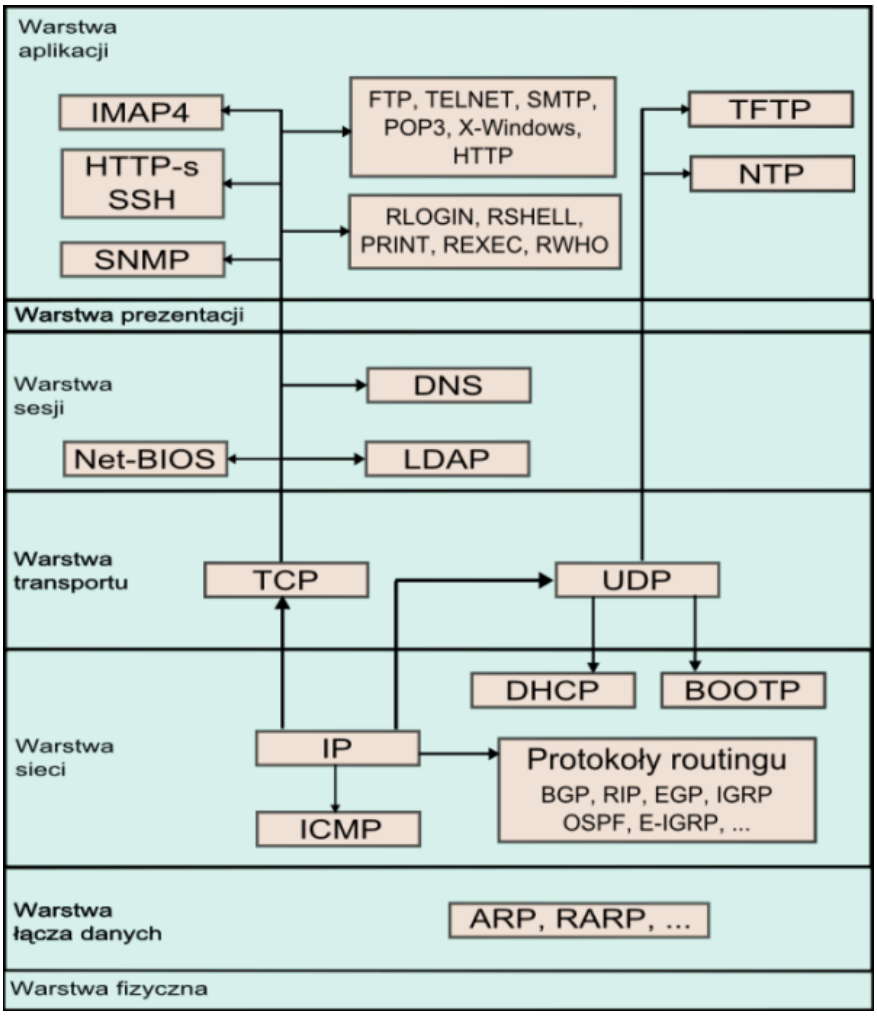


Porównanie modelu OSI i modelu TCP/IP.

Model OSI bywa wygodny dydaktycznie, bo rozkłada zagadnienia na siedem warstw, natomiast w świecie Internetu najczęściej myśli się „językiem TCP/IP", ponieważ to ten stos protokołów jest realnie zaimplementowany w systemach i urządzeniach sieciowych.

Rodzina protokołów TCP/IP

Mapa wybranych współczesnych protokołów, wraz ze wskazaniem, w jakiej warstwie modelu OSI one operują, znajduje się na poniższej ilustracji:



Mapa wybranych protokołów naniesionych na warstwy modelu OSI.

Aby TCP/IP mogło zapewnić jednolity podsystem komunikacyjny z punktu widzenia aplikacji, niezbędne jest spełnienie kilku kluczowych postulatów:

- Należy określić sposób adresowania. Adresy powinny być globalnie unikalne oraz – co równie ważne – routowalne w intersieci. Routowalne oznacza, że w adresie musi być zawarta informacja ułatwiająca wyznaczanie drogi, tak aby routery mogły podejmować decyzje gdzie przekazać pakiet dalej. Dla analogii: w adresie pocztowym taką „informacją do routingu" jest kod pocztowy – na jego podstawie przesyłka trafia między sortowniami i urzędami, a reszta adresu interesuje dopiero dostawcę na miejscu.

- Należy określić sposób wyznaczania trasy w intersieci. Trasa powinna być możliwie optymalna (lub przynajmniej sensowna), a decyzje mają być podejmowane rozproszenie: każdy router wybiera kolejne przekazanie pakietu w oparciu o informacje, które posiada lokalnie.
- W Internecie muszą działać mechanizmy pozwalające na autokonfigurację i skalowanie. Działanie sieci globalnej nie może polegać na ręcznych, centralnych działaniach pojedynczego operatora lub administratora.

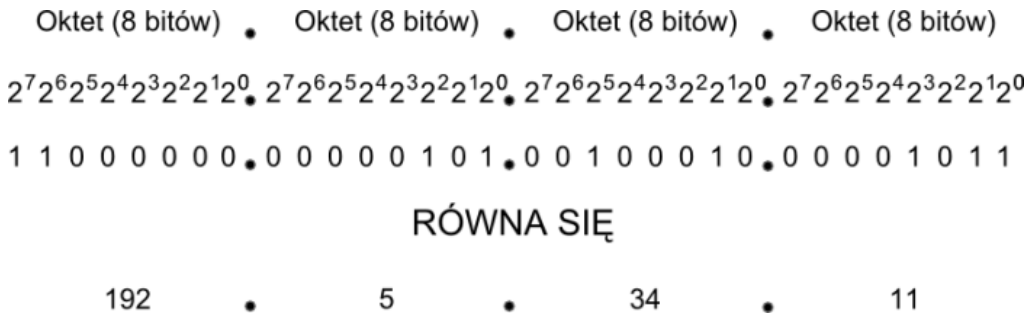
Opis całości zaczniemy więc od sposobu adresowania w Internecie. Następnie przejdziemy do zagadnień związanych z wyznaczaniem drogi (routingiem), po czym wrócimy do warstwy transportu: różnic między TCP i UDP, idei połączenia, niezawodności, kontroli przepływu oraz konsekwencji, jakie te mechanizmy mają dla działania aplikacji sieciowych.

1.2. Adresy IP

Żeby komputery mogły komunikować się w Internecie (i w sieciach lokalnych), muszą dać się jednoznacznie zidentyfikować w warstwie sieci. W rodzinie TCP/IP tę rolę pełni adres IP. W tym rozdziale skupimy się na adresowaniu w IPv4 (adres 32-bitowy), a na końcu omówimy też krótko IPv6.

Adres IPv4 i zapis dziesiętny z kropkami

W protokole IPv4 adres jest liczbą 32-bitową. Dla człowieka byłaby ona nieczytelna w postaci binarnej, dlatego powszechnie stosuje się zapis bajtowo-dziesiętny (ang. *dotted decimal*, czasem też *dotted quad notation*): adres dzieli się na cztery oktety (po 8 bitów), a każdy oktet zapisuje jako liczbę 0–255, rozdzielając kropkami.

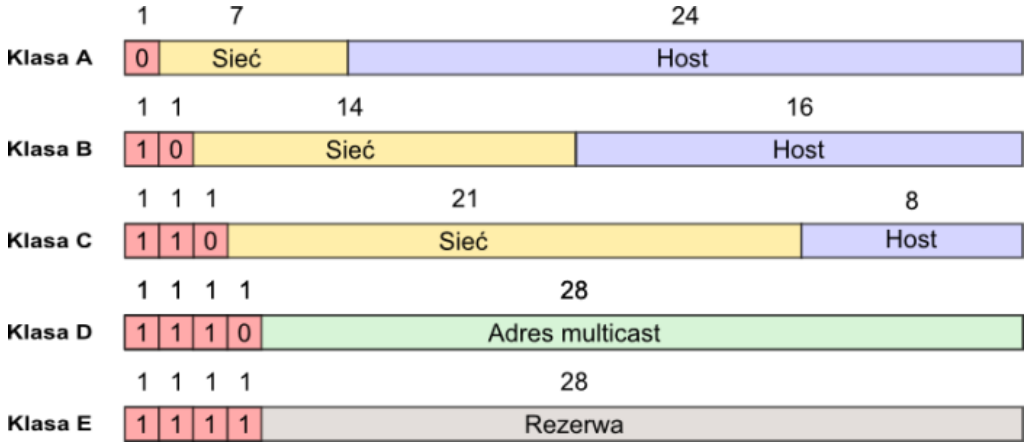


Zasada przedstawiania adresu IPv4 (32 bity) w zapisie dziesiętnym z kropkami.

Adres IP nie jest tylko „numerkiem hosta". W klasycznym podejściu zakładano, że zawsze da się go podzielić na: część identyfikującą sieć oraz część identyfikującą hosta w tej sieci. Współcześnie ten podział nadal istnieje, ale jest realizowany inaczej niż dawniej (patrz CIDR i prefiksy).

Adresowanie klasowe jako kontekst historyczny

W początkach Internetu stosowano tzw. adresowanie klasowe (A, B, C). Klasa adresu wynikała z kilku pierwszych bitów i narzucała „sztywną" granicę podziału na część sieciową i hostową. Dziś to podejście traktujemy głównie jako kontekst historyczny: pomaga zrozumieć, skąd wzięły się skróty myślowe typu „/8, /16, /24", ale nie jest już podstawą routingu w Internecie.



Klasyczny (historyczny) podział adresów IPv4 na klasy.

Klasa A

Adres klasy A zaczyna się od bitu 0. Historycznie oznaczało to bardzo duże sieci (ogromna liczba hostów w jednej sieci). W praktyce takie „płaskie" sieci byłyby nieefektywne i trudne w utrzymaniu, dlatego nawet jeśli ktoś miał przydział klasy A, to i tak dzielił go na mniejsze podsieci.

Klasa B

Adres klasy B zaczyna się od bitów 10 i był projektowany dla „średnich i dużych" sieci (więcej sieci niż w klasie A, ale mniej hostów w pojedynczej sieci).

Klasa C

Adres klasy C zaczyna się od bitów 110 i był projektowany dla wielu małych sieci (bardzo dużo sieci, mało hostów w każdej z nich).

Klasy D i E

Klasa D (224.0.0.0–239.255.255.255) to adresy multicast. Klasa E (240.0.0.0–255.255.255.255) była rezerwowana na cele eksperymentalne. Te zakresy nie służą do klasycznego adresowania hostów w Internecie.

Routing bezklasowy (CIDR), prefiks i maska podsieci

Adresowanie klasowe prowadziło do marnowania ogromnych puli adresów (np. przydział „zbyt dużej” sieci). Dlatego wprowadzono routing bezklasowy CIDR (ang. *Classless Inter-Domain Routing*), w którym granica między częścią sieci i hosta nie wynika z klasy, tylko z **prefiksu** (długości części sieciowej).

Prefiks zapisujemy jako **/n**, gdzie *n* to liczba bitów części sieciowej, np.: **192.168.1.0/24**. Oznacza to, że pierwsze 24 bity identyfikują sieć, a pozostałe 8 bitów identyfikuje hosta. Równoważnym zapisem jest **maska podsieci** (ang. *subnet mask*), czyli 32-bitowa liczba, w której bity części sieciowej są jedynkami, a bity części hosta zerami. Dla /24 maska to **255.255.255.0**.

CIDR pozwala tworzyć podsieci dokładnie dopasowane rozmiarem do potrzeb, np. /26 (64 adresy, z czego typowo 62 dla hostów), /28 (16 adresów, typowo 14 dla hostów) itd. Dzięki temu adresy IPv4 są wykorzystywane efektywniej, a routing w Internecie może agregować prefiksy (mniej wpisów w tablicach routingu).

Adresy specjalnego przeznaczenia

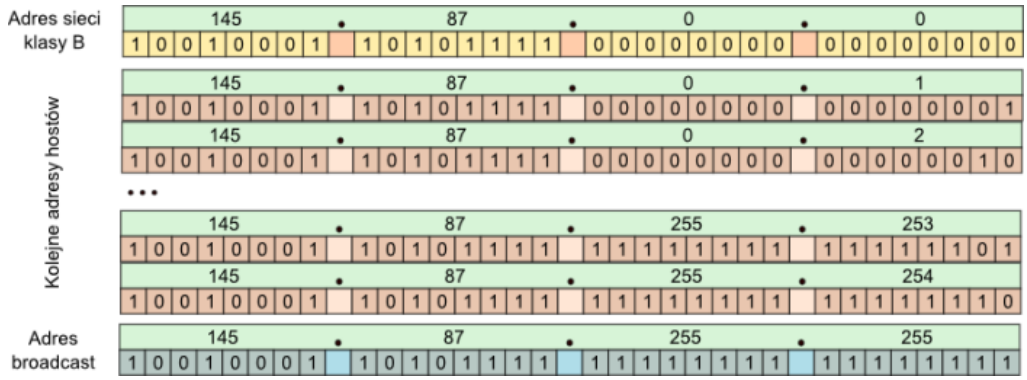
Nie wszystkie adresy IPv4 mogą być przypisywane hostom. Część ma znaczenie specjalne na poziomie protokołu lub routingu. Poniżej najważniejsze przypadki, z którymi spotkasz się w praktyce.

Adres sieci i adres rozgłoszeniowy (broadcast)

Dla danego prefiksu (np. /24) istnieją dwa adresy, których nie przypisuje się hostom:

- **adres sieci** – wszystkie bity części hosta są zerami (np. 192.168.1.0/24),
- **adres rozgłoszeniowy** – wszystkie bity części hosta są jedynkami (np. 192.168.1.255/24).

Adres rozgłoszeniowy służy do wysyłania pakietów do wszystkich hostów w danej sieci/podsieci (broadcast lokalny). W praktyce broadcast jest używany m.in. w procesach konfiguracji i odkrywania usług w sieci lokalnej.



Przykład wykorzystania adresu sieci i adresu rozgłoszeniowego w podsieci.

0.0.0.0 i trasa domyślna

Adres **0.0.0.0** jest adresem „nieokreślonym” (host używa go np. zanim pozna własny adres). W tablicach routingu zapis **0.0.0.0/0** oznacza **trasę domyślną** (prefiks długości 0 pasuje do wszystkiego), czyli „jeśli nie masz bardziej szczegółowej trasy, wyślij pakiet tutaj”.

Pętla zwrotna (loopback)

Zakres **127.0.0.0/8** jest zarezerwowany jako pętla zwrotna (*loopback*). Najczęściej używa się adresu **127.0.0.1** (nazwa symboliczna: **localhost**). Pakiety wysłane na loopback nie wychodzą na sieć fizyczną: wracają do nadawcy wewnątrz systemu operacyjnego. To mechanizm diagnostyczny, ale też popularny sposób komunikacji między procesami na tym samym komputerze.

Adresy prywatne (RFC 1918)

W sieciach lokalnych (dom, SOHO, firma) bardzo często stosuje się **adresy prywatne**, czyli pule IPv4 nieprzeznaczone do routowania w publicznym Internecie. Routery w Internecie nie powinny przekazywać pakietów z adresami prywatnymi, dlatego takie adresy są „bezpieczne” do wielokrotnego użycia w wielu niezależnych sieciach.

Zakresy adresów prywatnych (RFC 1918) to:

- **10.0.0.0 – 10.255.255.255** (10.0.0.0/8),
- **172.16.0.0 – 172.31.255.255** (172.16.0.0/12),
- **192.168.0.0 – 192.168.255.255** (192.168.0.0/16).

Wybór zakresu i podział na podsieci (prefiksy) zależy od administratora. W praktyce domowe routery SOHO często konfigurują sieć jako 192.168.0.0/24 lub 192.168.1.0/24, ale nie jest to wymóg.

NAT: dlaczego powstał i co rozwiązuje

Adresy publiczne IPv4 są dobrem ograniczonym. Wraz ze wzrostem liczby urządzeń podłączanych do sieci (komputery, smartfony, IoT) stało się jasne, że IPv4 nie zapewni „po jednym publicznym adresie na urządzenie". Jednym z praktycznych obejść problemu jest **NAT** (ang. *Network Address Translation*) – translacja adresów na granicy sieci prywatnej i publicznej.

Najczęstszy scenariusz SOHO wygląda tak: wiele hostów w sieci prywatnej (np. 192.168.1.0/24) korzysta z Internetu przez jeden (lub niewiele) publiczny adres IP na routerze. NAT zmienia adresy w pakietach, tak aby ruch z sieci prywatnej mógł poprawnie działać w sieci publicznej.

Terminologia NAT: „inside/outside" oraz „local/global"

Żeby precyzyjnie opisywać NAT, wygodnie jest rozróżnić dwie osie pojęć:

- **inside** (wewnątrz) – sieć prywatna za NAT,
- **outside** (na zewnątrz) – Internet / sieć publiczna.

Oraz:

- **local** – adres widoczny „lokalnie" w danej strefie (inside lub outside),
- **global** – adres widoczny „globalnie" po drugiej stronie NAT (po translacji).

W efekcie można spotkać cztery pojęcia (w praktyce najczęściej potrzebne są dwa pierwsze):

- **inside local** – prywatny adres hosta w sieci lokalnej (np. 192.168.1.10),
- **inside global** – publiczny adres, pod którym ten host „wychodzi" do Internetu (często adres routera),
- **outside global** – publiczny adres serwera w Internecie (np. adres strony WWW),
- **outside local** – adres serwera widziany „od środka" (zwykle taki sam jak outside global, ale w pewnych konfiguracjach może być mapowany).

Rodzaje NAT: statyczny, dynamiczny oraz NAPT/PAT

NAT statyczny (1:1)

NAT statyczny mapuje jeden adres prywatny na jeden adres publiczny. Stosuje się go wtedy, gdy chcemy, aby konkretny host „miał swój" publiczny adres (np. serwer w firmie), ale nadal był fizycznie w sieci prywatnej.

NAT dynamiczny (pula publicznych adresów)

NAT dynamiczny korzysta z puli publicznych adresów i przydziela je hostom tymczasowo – w miarę potrzeb. To rozwiązanie oszczędza adresy w porównaniu do 1:1, ale nadal wymaga posiadania wielu adresów publicznych.

NAPT/PAT (wiele hostów na jeden adres publiczny)

Najczęściej spotykane w domach i SOHO jest rozwiązanie, w którym wiele hostów współdzieli jeden publiczny adres IP dzięki wykorzystaniu portów TCP/UDP. To podejście nazywa się zwykle: **NAPT** (Network Address and Port Translation) albo **PAT** (Port Address Translation). Router zmienia nie tylko adres IP, ale też numer portu źródłowego, tworząc unikalne mapowanie połączeń.

NAT w praktyce: port forwarding i konsekwencje dla serwerów

Często spotyka się uproszczenie: „host za NAT nie może być serwerem". To nie jest do końca prawda. Host za NAT **może** udostępniać usługę na zewnątrz, ale zwykle wymaga to jawnej konfiguracji na routerze: **przekierowania portów** (ang. *port forwarding*) albo mapowania 1:1.

Jeśli router nie ma skonfigurowanego mapowania ruchu przychodzącego, to nie wie, do którego hosta w sieci wewnętrznej przekazać pakiet „z Internetu". Właśnie dlatego NAT dobrze działa dla połączeń inicjowanych od środka (inside → outside), a gorzej dla połączeń inicjowanych z zewnątrz (outside → inside).

Zalety i wady NAT

Zalety:

- oszczędza publiczne adresy IPv4 (kluczowy powód jego popularności),
- upraszcza adresowanie w sieciach lokalnych (można używać adresów prywatnych),
- działa jako „efekt uboczny" ograniczający część niechcianego ruchu przychodzącego (ale nie zastępuje firewalla).

Wady i problemy praktyczne:

- utrudnia bezpośrednie połączenia przychodzące (serwery, zdalny dostęp, gry),
- komplikuje część protokołów i aplikacji (VoIP, P2P),
- wymusza dodatkowe mechanizmy obejścia w połączeniach P2P (np. „dziurkowanie" NAT).

> ## Komentarz
>
> W aplikacjach P2P często stosuje się techniki typu *UDP hole punching* (w uproszczeniu: oba hosty inicjują ruch „na zewnątrz”, a serwer pośredni pomaga im zsynchronizować mapowania w NAT, by mogły wymieniać pakiety bez stałego pośrednika). To temat praktyczny i ważny, ale wykracza poza podstawy adresowania.

IP w wersji 6

IPv6 wprowadza adresy 128-bitowe, co daje ogromną przestrzeń adresową (2^{128} możliwych wartości). Dzięki temu problem deficytu adresów, który w IPv4 „łata się” m.in. NAT-em, w IPv6 jest rozwiązany na poziomie samego protokołu.

Zapis adresu IPv6 wykorzystuje zapis szesnastkowy i dwukropki co 16 bitów, np.: `b091:2120:a300:0000:0201:021f:e234:2001`. Można pomijać wiodące zera w blokach, a ciąg kolejnych bloków zer skrócić do `::` (jednorazowo w adresie), np. `b091:2120:a300::201:21f:e234:2001`.

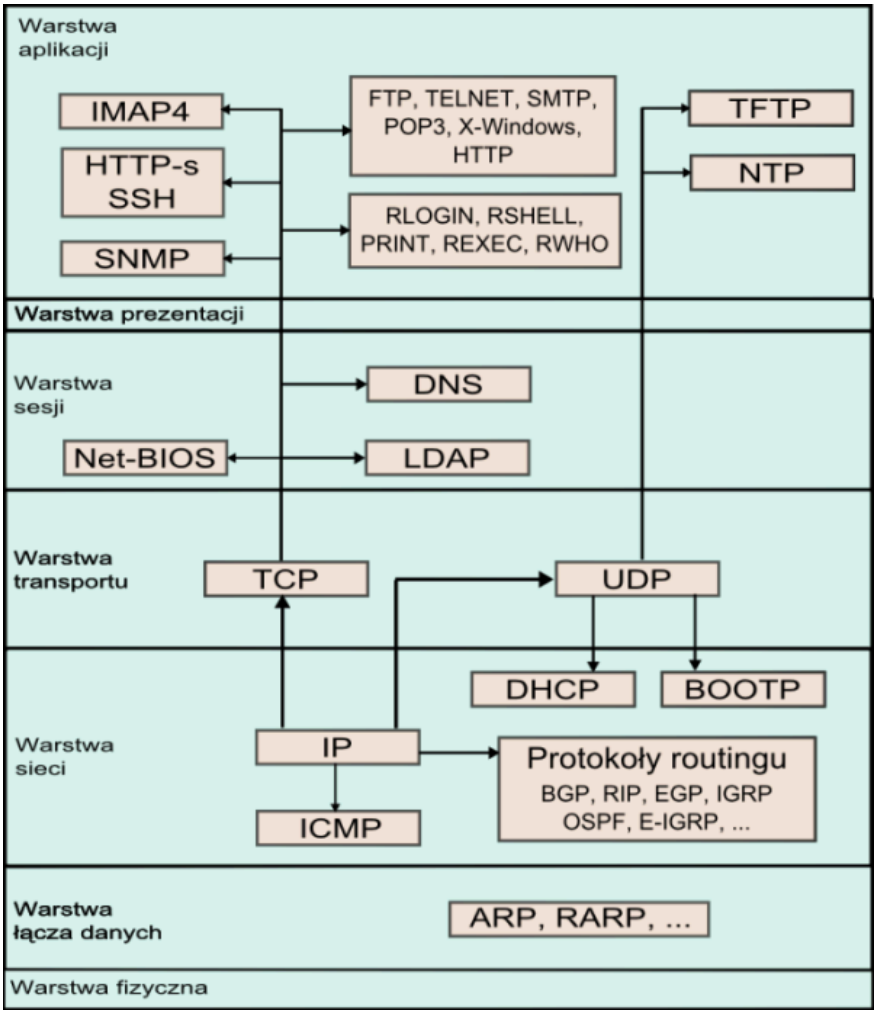
W IPv6 również powszechnie używa się prefiksów (CIDR), np. `/64` dla typowych sieci LAN. Szczegóły (autokonfiguracja, rodzaje adresów, sąsiedztwo, itp.) omówimy w kolejnych rozdziałach, bo IPv6 to nie tylko „dłuższy adres”, ale też zestaw zmian porządkujących działanie protokołu.

W następnym rozdziale przejdziemy do praktycznej interpretacji adresu jako „prefiks + host”, czyli do masek/prefiksów, podsieci i tego, jak routery wykorzystują adres do wyznaczania drogi w intersieci.

1.3. Opis podstawowych protokołów

Najważniejsze protokoły omówimy w podziale na warstwy modelu OSI, pamiętając jednocześnie, że w praktyce (w rodzinie TCP/IP) granice między warstwami bywają „pragmatyczne” – liczy się przede wszystkim to, jakie zadanie dany protokół realizuje w całym stosie.

Przypomnijmy jeszcze raz mapę wybranych protokołów:



Mapa wybranych protokołów naniesionych na warstwy modelu OSI.

Warstwa łącza danych

W tej warstwie (a dokładniej: „na styku” warstwy łącza danych i warstwy sieci) znajdują się mechanizmy, które pozwalają hostom ustalić, jaki adres fizyczny (MAC) odpowiada konkretnemu adresowi IP w ramach jednej sieci lokalnej.

ARP (ang. *Address Resolution Protocol*) służy w sieciach **IPv4** do odwzorowania adresu IP na adres MAC. Jest to konieczne, ponieważ w typowej sieci LAN ramki warstwy 2 są kierowane według adresów MAC, a nie według adresów IP. Działanie ARP jest rozgłoszeniowe: host wysyła zapytanie „kto ma IP X?” do wszystkich w danej domenie rozgłoszeniowej, odpowiada tylko urządzenie posiadające wskazany adres IP. Warto podkreślić, że ARP działa tylko lokalnie – router nie przenosi zapytań ARP do innych sieci.

[Protokół ARP]
Działanie protokołu ARP.

RARP (ang. *Reverse Address Resolution Protocol*) był historycznym „odwróceniem” ARP: host znający swój adres MAC próbował uzyskać odpowiadający mu adres IP. W praktyce RARP jest dziś protokołem **niemal niespotykanym**, a jego rolę przejęły rozwiązania konfiguracyjne takie jak BOOTP/DHCP (oraz inne mechanizmy autokonfiguracji). Jeśli spotkacie RARP w literaturze, traktujcie go raczej jako element historii rozwoju TCP/IP.

Uwaga o IPv6: w sieciach IPv6 nie stosuje się ARP. Jego rolę pełni **NDP** (ang. *Neighbor Discovery Protocol*), będący częścią **ICMPv6**. NDP również pozwala ustalić „kto jest kim” w sieci lokalnej, ale robi to w sposób dostosowany do IPv6.

Warstwa sieci

W tej warstwie kluczowy jest protokół **IP** (ang. *Internet Protocol*) – fundament routowania i przesyłania datagramów między sieciami.

IP odpowiada za przeniesienie datagramu od nadawcy do odbiorcy przez potencjalnie wiele sieci pośrednich i wiele routerów. Protokół IP jest „bezstanowy” – kolejne datagramy tej samej transmisji mogą iść różnymi trasami, mogą dotrzeć w innej kolejności, a część może zaginąć. IP nie gwarantuje dostarczenia – dostarcza jedynie mechanizm adresowania i przekazywania pakietów oraz minimalną kontrolę poprawności nagłówka.

ICMP (ang. *Internet Control Message Protocol*) to protokół kontrolny i diagnostyczny współpracujący z IP. Służy do przesyłania komunikatów o błędach i do testowania łączności. Najbardziej znanym zastosowaniem ICMP jest **ping**, który używa komunikatów Echo Request / Echo Reply do sprawdzenia, czy host odpowiada oraz jaki jest przybliżony czas odpowiedzi.

DHCP (ang. *Dynamic Host Configuration Protocol*) umożliwia automatyczne uzyskanie konfiguracji sieciowej: adresu IP, maski/prefiksu, bramy domyślnej, adresów DNS i innych parametrów. Warto doprecyzować warstwowo: DHCP jest protokołem **konfiguracyjnym** działającym na wyższym poziomie i zwykle korzysta z **UDP**, a na starcie często posługuje się rozgłoszeniami, bo host nie zna jeszcze swoich parametrów i nie potrafi kierować pakietów „normalnie”.

BOOTP (ang. *BOOTstrap Protocol*) jest starszym i prostszym poprzednikiem DHCP. Współcześnie spotyka się go głównie w zastosowaniach specjalnych (np. rozruch sieciowy, niektóre środowiska wbudowane), ale w typowych sieciach SOHO i firmowych jego rolę przejął DHCP.

Protokoły routingu

Routing (wyznaczanie drogi) w praktyce opiera się na tablicach routingu routerów. Taka tablica zawiera informacje o sieciach docelowych (prefiksach), o „następnym skoku” (next-hop) lub interfejsie wyjściowym oraz o metryce (koszcie). Trasy mogą być wprowadzone ręcznie (routing statyczny) albo pozyskiwane automatycznie (routing dynamiczny).

W terminologii spotkacie dwa podobne, ale różne pojęcia:

- **protokół routowalny** – protokół, którego adresowanie pozwala przenosić pakiety przez wiele sieci i węzłów pośrednich (np. **IP**);
- **protokół routingu** – protokół wymiany informacji o trasach pomiędzy routerami, dzięki któremu tablice routingu są aktualizowane automatycznie.

Przykłady protokołów routingu: **RIP**, **OSPF**, **IS-IS**, **IGRP** oraz **EIGRP**. W bardzo dużych sieciach i między operatorami kluczową rolę odgrywa **BGP** (protokół routingu „międzydomenowego”). Szczegółowe algorytmy (metryki, zbieżność, konwergencja po awarii) wykraczają poza ten rozdział, ale warto zapamiętać: dynamiczny routing to mechanizm, który pozwala sieci „adaptować się” do zmian bez ręcznego przepisywania tras w każdym routerze.

RIP jako przykład protokołu routingu

Do tej pory mówiliśmy o tym, *jak* opisać sieci i trasy (prefiksy, maski, CIDR), ale pozostaje pytanie: skąd router „wie”, dokąd wysłać pakiet? Odpowiedzią są tablice routingu, które mogą być budowane ręcznie (trasy statyczne) albo automatycznie przez **protokoły routingu**. Jednym z najprostszych i historycznie bardzo popularnych protokołów jest **RIP** (ang. *Routing Information Protocol*), zaliczany do protokołów routingu typu **distance-vector** (wektor odległości).

RIP działa w oparciu o prostą metrykę: **liczbę przeskoków** (ang. *hop count*). Każdy router okresowo rozsyła do sąsiadów swoją tabelę routingu (w praktyce: listę znanych prefiksów wraz z odległością). Gdy router odbierze taką informację, „dodaje” jeden przeskok i porównuje wynik z tym, co ma w swojej tablicy: jeśli nowa droga jest krótsza, aktualizuje wpis i zapamiętuje, przez którego sąsiada należy wysyłać pakiety. Dzięki temu informacje o dostępnych sieciach „rozlewają się” po całej infrastrukturze bez centralnego sterowania.

Prostota RIP ma jednak cenę. Po pierwsze, protokół ma ograniczenie metryki: **maksymalnie 15 przeskoków** (16 oznacza „nieosiągalne”), więc nie nadaje się do dużych sieci. Po drugie, zbieżność (czas dochodzenia do stabilnego stanu po awarii) bywa powolna, a w pewnych scenariuszach może dochodzić do problemów typu „zliczanie do nieskończoności” (gdy routery długo „wierzą” w nieaktualne informacje). Dlatego RIP spotyka się dziś głównie w małych, prostych sieciach lub jako przykład dydaktyczny, a w większych środowiskach stosuje się nowsze protokoły (np. OSPF/IS-IS wewnątrz organizacji czy BGP między operatorami), o których warto pamiętać jako o kolejnych krokach rozwoju routingu.

Warstwa transportu

TCP (ang. *Transmission Control Protocol*) to protokół połączeniowy i niezawodny. Zapewnia uporządkowane dostarczenie danych, wykrywanie braków oraz retransmisję. „Połączeniowość” w TCP oznacza ustanowienie **logicznego** kanału komunikacji (wirtualnego połączenia) oraz utrzymywanie stanu po obu stronach (numery sekwencyjne, potwierdzenia, okno transmisji). Nie oznacza to istnienia stałego fizycznego obwodu – w sieci IP pakiety mogą wędrować różnymi trasami, a TCP dba o to, aby aplikacja po stronie odbiorcy widziała spójny strumień bajtów.

UDP (ang. *User Datagram Protocol*) jest protokołem bezpołączeniowym i „niezawodnościowo neutralnym”: nie gwarantuje dostarczenia, kolejności ani braku duplikatów. Zapewnia natomiast prosty mechanizm portów i niewielki narzut, dzięki czemu bywa bardzo użyteczny tam, gdzie liczy się szybkość, niskie opóźnienia albo gdzie niezawodność realizuje się na wyższym poziomie (np. w samej aplikacji).

Pojęcie portu

TCP i UDP, oprócz adresów IP, używają numerów portów źródłowego i docelowego. Port identyfikuje konkretną usługę lub proces na hoście. Dzięki temu jeden komputer może równolegle obsługiwać wiele połączeń i wiele usług sieciowych.

Numery portów należą do zakresu 0–65535 (16 bitów). W praktyce stosuje się następujący podział:

- **0–1023** – porty ogólnie znane (ang. *well-known*), typowo używane przez standardowe usługi;
- **1024–49151** – porty zarejestrowane (ang. *registered*), często przypisane aplikacjom/produktom;
- **49152–65535** – porty dynamiczne/efemeryczne (ang. *dynamic/ephemeral*), zwykle wybierane automatycznie dla połączeń wychodzących.

Przykłady popularnych portów: 21 – FTP, 22 – SSH, 53 – DNS, 80 – HTTP, 443 – HTTPS.

Protokoły wyższych warstw

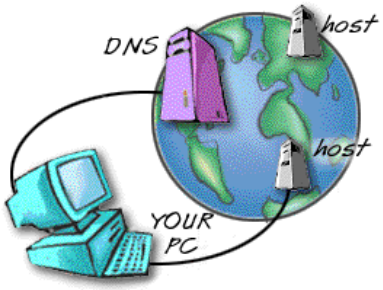
Protokoły wyższych warstw realizują konkretne usługi: pocztę (SMTP/IMAP/POP3), tłumaczenie nazw (DNS), transfer plików (FTP/TFTP), zdalny dostęp (SSH) i wiele innych. Działają „na” TCP lub UDP i korzystają z mechanizmów adresowania, routingu i transportu opisanych powyżej.

1.4. Protokoły i usługi stowarzyszone

W praktycznej komunikacji sieciowej rola IP oraz protokołów transportowych (TCP i UDP) polega na dostarczeniu „nośnika” dla danych aplikacji: albo w postaci niezawodnego strumienia bajtów (TCP), albo w postaci lekkich datagramów (UDP). Dopiero na tym fundamencie buduje się protokoły wyższych warstw, zwykle zaliczane do warstwy aplikacji, które realizują konkretne usługi: tłumaczenie nazw (DNS), pocztę elektroniczną, zarządzanie infrastrukturą, zdalny dostęp, synchronizację czasu i wiele innych.

DNS

Pamiętanie adresów IP w formie liczbowej nie jest wygodne. Ludzie wolą posługiwać się nazwami opisowymi, natomiast routery muszą pracować na adresach IP. Rozwiązaniem jest **DNS** (ang. *Domain Name System*) – system tłumaczenia nazw domenowych na adresy IP (i w pewnym zakresie również odwrotnie).



System nazw domenowych – DNS.

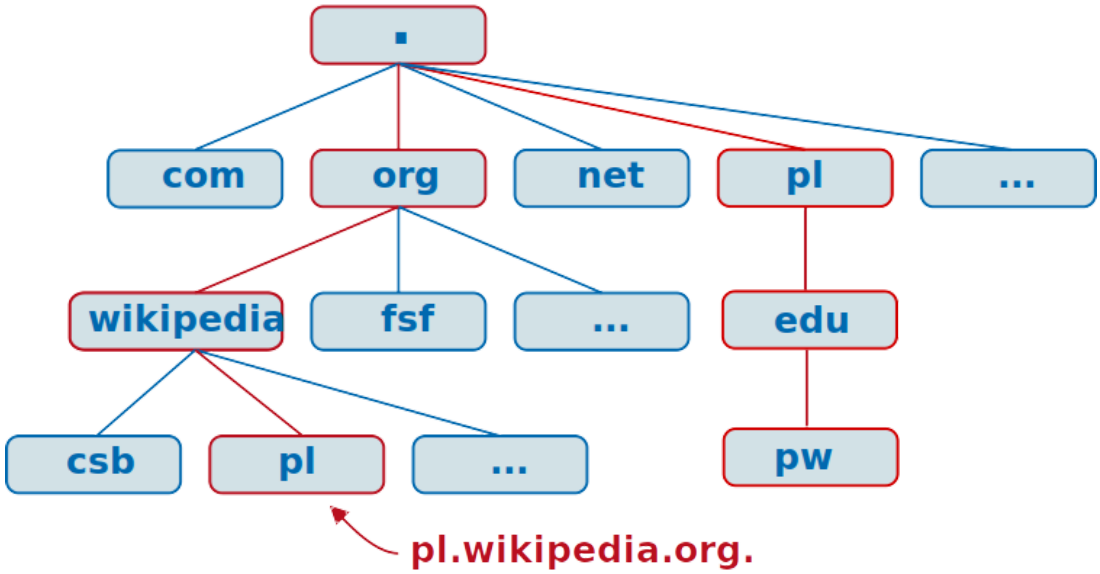
Nazwa domenowa to hierarchiczny ciąg etykiet oddzielonych kropkami (np.

```
<a href="<a href="http://www.przyklad.pl">Oferta sprzedaży domeny przyklad.pl</a>" class="_blanktarget"><a
href="http://www.przyklad.pl">Oferta sprzedaży domeny przyklad.pl</a></a>
```

). DNS jest systemem rozproszonym i delegacyjnym: nie istnieje jedna centralna „książka adresowa” Internetu. Zamiast tego odpowiedzialność za fragmenty drzewa nazw (strefy) jest delegowana kolejnym podmiotom, a serwery DNS przechowują informacje tylko o tych strefach, za które odpowiadają.

W uproszczeniu w procesie rozwiązywania nazw biorą udział dwa typy elementów:

- **resolver** (zwykle rekurencyjny) – komponent po stronie klienta/u dostawcy Internetu, który „załatwia” wyszukiwanie odpowiedzi w hierarchii DNS i buforuje wyniki;
- **serwer autorytatywny** – serwer, który posiada wiarygodne dane dla danej strefy (np. dla domeny `przyklad.pl`).



Uproszczony schemat działania DNS.

Górny poziom hierarchii obejmuje strefę root oraz domeny najwyższego poziomu (TLD), np. `pl`, `com`, `org`, `edu`. Następnie występują domeny niższych poziomów, które mogą być delegowane dalej. W praktyce to, kto zarządza danym fragmentem drzewa, wynika z delegacji (rekordów typu NS) oraz konfiguracji stref.

W DNS przechowuje się różne typy rekordów. Najczęściej spotkacie:

- **A** – mapowanie nazwy na adres IPv4,
- **AAAA** – mapowanie nazwy na adres IPv6,

- **CNAME** – alias (nazwa wskazuje na inną nazwę),
- **MX** – serwery obsługujące pocztę dla domeny,
- **NS** – serwery nazw odpowiedzialne za strefę,
- **TXT** – rekord tekstowy (często wykorzystywany m.in. w konfiguracjach bezpieczeństwa poczty).

Ważną cechą DNS jest buforowanie (cache). Odpowiedzi mogą być przechowywane przez resolver przez czas określony parametrem **TTL** (ang. *Time To Live*). Przyspiesza to działanie całego systemu i zmniejsza liczbę zapytań, ale powoduje też, że zmiany w rekordach DNS mogą „rozchodzić się" w Internecie z opóźnieniem.

Istnieje także mechanizm wzmacniający wiarygodność odpowiedzi – **DNSSEC** – który pozwala kryptograficznie weryfikować integralność danych DNS. Nie jest to szyfrowanie zapytań, lecz mechanizm podpisu i weryfikacji odpowiedzi.

Warto pamiętać, że „odwrotne" mapowanie IP → nazwa (tzw. reverse DNS) istnieje, ale nie jest pełnym i jednoznacznym odwróceniem: jeden adres IP może obsługiwać wiele nazw (wirtualny hosting), a rekord reverse DNS (PTR) jest dodatkową informacją konfiguracyjną, a nie wynikiem „automatycznego odwrócenia" DNS.

LDAP

LDAP (ang. *Lightweight Directory Access Protocol*) jest protokołem dostępu do usług katalogowych. Katalog LDAP przechowuje dane w strukturze drzewa, nastawionej na szybkie wyszukiwanie i odczyt (to nie jest relacyjna baza danych). Dane są przechowywane w postaci wpisów (Entries), a każdy wpis ma zestaw atrybutów zgodnych ze zdefiniowanym schematem.

W katalogach LDAP często spotkacie pojęcia takie jak:

- **DN** (Distinguished Name) – pełna, unikalna nazwa wpisu w drzewie,
- **OU** (Organizational Unit) – jednostka organizacyjna, często używana do porządkowania struktury,
- **ACL** – listy kontroli dostępu określające uprawnienia do odczytu i modyfikacji.

LDAP bywa wykorzystywany do przechowywania informacji o użytkownikach, grupach, urządzeniach i uprawnieniach. Popularne implementacje to **Active Directory** firmy Microsoft oraz **OpenLDAP**. W praktyce LDAP bardzo często jest elementem systemu logowania i zarządzania tożsamością w organizacji.

SNMP

W sieciach komputerowych potrzebujemy monitorowania i zarządzania urządzeniami: przełącznikami, routerami, punktami dostępowymi, serwerami czy zasilaniem awaryjnym. Jednym z klasycznych protokołów do takich zastosowań jest **SNMP** (ang. *Simple Network Management Protocol*).

SNMP opiera się na pojęciu **MIB** (ang. *Management Information Base*) – jest to uporządkowany zbiór obiektów/parametrów opisujących stan urządzenia (np. obciążenie interfejsu, licznik błędów, temperatury). System zarządzający (manager) może odczytywać wartości (np. operacje typu GET), a w pewnych przypadkach także je ustawiać (SET). Urządzenia mogą też wysyłać asynchroniczne powiadomienia o zdarzeniach (TRAP/INFORM).

Warto pamiętać o bezpieczeństwie: starsze wersje SNMP (v1, v2c) używają tzw. „community string" i nie zapewniają silnych mechanizmów ochrony. Dopiero **SNMPv3** wprowadza uwierzytelnianie i (opcjonalnie) szyfrowanie, dlatego we współczesnych wdrożeniach to właśnie v3 jest pożądanym wyborem.

POP3, SMTP, IMAP4

SMTP (ang. *Simple Mail Transfer Protocol*) służy do przekazywania poczty między serwerami oraz do wysyłania wiadomości z klienta do serwera. **POP3** (ang. *Post Office Protocol* v3) i **IMAP4** (ang. *Internet Message Access Protocol* v4) są używane przez aplikacje klienckie do odbioru poczty z serwera.

Różnica praktyczna między POP3 i IMAP4 jest taka, że POP3 tradycyjnie nastawiony jest na pobranie wiadomości „na komputer", natomiast IMAP4 zakłada pracę na skrzynce przechowywanej na serwerze (foldery, etykiety, synchronizacja wielu urządzeń). Dziś, przy smartfonach i wielu klientach jednocześnie, IMAP jest rozwiązaniem bardzo popularnym.

We współczesnej poczcie istotne są mechanizmy bezpieczeństwa: szyfrowanie połączeń (TLS), uwierzytelnianie nadawcy (SMTP AUTH) oraz polityki antyspamowe i reputacyjne. Z tego powodu uruchomienie „własnego serwera pocztowego" jest możliwe, ale wymaga poprawnej konfiguracji (m.in. rekordów DNS i polityk) – w przeciwnym razie wiadomości mogą trafiać do spamu lub być odrzucane.

HTTP i HTTPS

HTTP (ang. *Hypertext Transfer Protocol*) to podstawowy protokół sieci WWW. Umożliwia pobieranie dokumentów (HTML), obrazów, skryptów, danych API i wielu innych zasobów. Z punktu widzenia praktycznego HTTP jest dziś uniwersalnym „transportem" dla usług sieciowych – nie tylko dla stron internetowych.

Przeglądarka internetowa renderuje (odtwarza) dokument HTML i zasoby powiązane, umożliwiając nawigację dzięki hiperłączom. Współcześnie WWW to nie tylko statyczne strony – to również aplikacje webowe i interfejsy API, które komunikują się z serwerami w sposób automatyczny.

HTTPS to HTTP uruchomione „w tunelu" szyfrowanym przez **TLS**. Zapewnia poufność i integralność transmisji oraz (w typowym scenariuszu) uwierzytelnienie serwera za pomocą certyfikatu. Dziś większość ruchu WWW odbywa się przez HTTPS.

Po stronie serwera spotkacie m.in. Apache, Nginx czy Microsoft IIS (ang. *Internet Information Services*). Po stronie klienta – przeglądarki takie jak Chrome, Firefox czy Edge. Istnieją też przeglądarki tekstowe (np. Lynx), użyteczne w środowiskach terminalowych.

TFTP i FTP

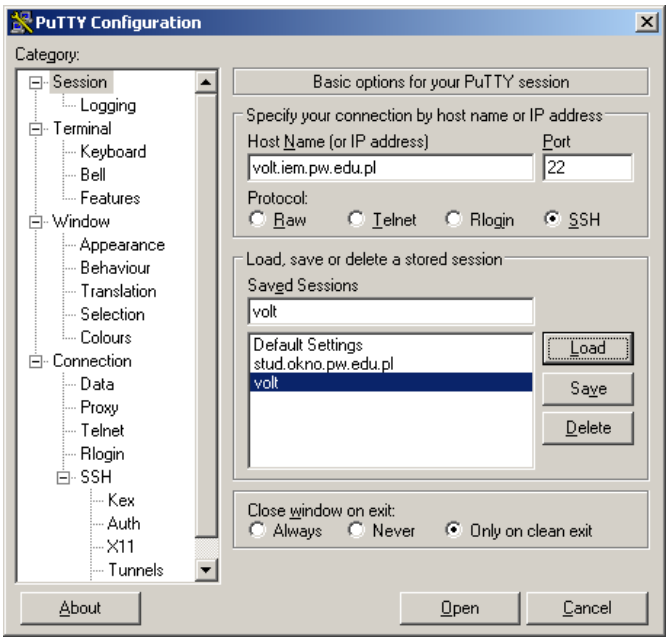
TFTP (ang. *Trivial File Transfer Protocol*) jest bardzo prostym protokołem przesyłania plików opartym o UDP. Z tego powodu ma niski narzut, ale nie zapewnia własnych mechanizmów niezawodności i bezpieczeństwa, więc używa się go zwykle w specyficznych zastosowaniach (np. rozruch sieciowy, proste urządzenia).

FTP (ang. *File Transfer Protocol*) to klasyczny protokół transferu plików działający na TCP. Ma bogatsze możliwości niż TFTP, ale w swojej podstawowej postaci nie szyfruje transmisji. W praktyce, w zastosowaniach wymagających bezpieczeństwa, FTP bywa zastępowany przez rozwiązania szyfrowane, takie jak **SFTP** (transfer plików przez SSH) albo przesyłanie plików przez HTTPS.

TELNET i SSH

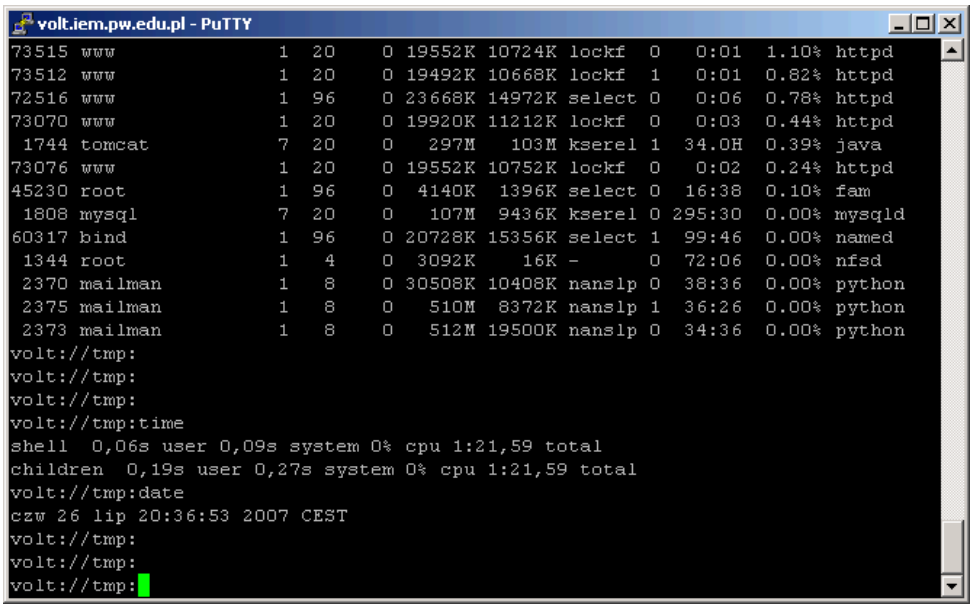
TELNET i **SSH** umożliwiają zdalną pracę w trybie tekstowym na innym komputerze. TELNET nie szyfruje transmisji, więc we współczesnych sieciach traktuje się go jako rozwiązanie przestarzałe i potencjalnie niebezpieczne. **SSH** (ang. *Secure Shell*) zapewnia szyfrowanie oraz mechanizmy uwierzytelniania, dlatego jest standardem w administracji systemami Unix/Linux, a także jest powszechnie używany również w środowiskach Windows.

W praktyce po stronie serwera działa usługa (demon) SSH, a po stronie klienta – program kliencki. Popularne narzędzia klienckie to m.in. PuTTy, OpenSSH czy WinSCP (do pracy z plikami). W środowiskach graficznych, jako odpowiednik „zdalnej pracy na ekranie", spotyka się protokoły typu RDP (Windows) lub VNC.



Ekran powitalny programu PuTTy.

PuTTy umożliwia skonfigurowanie sesji: wybór protokołu, adresu serwera oraz portu docelowego.



Otwarta sesja SSH z serwerem za pomocą programu PuTTy.

Po ustanowieniu połączenia i uwierzytelnieniu użytkownika uzyskujemy dostęp do powłoki systemu zdalnego, zgodnie z przyznanymi uprawnieniami.

NTP

NTP (ang. *Network Time Protocol*) służy do synchronizacji czasu pomiędzy komputerami. Choć może się wydawać mało istotny, poprawny czas bywa krytyczny dla analizy logów, korelacji zdarzeń bezpieczeństwa, a także dla działania wielu mechanizmów uwierzytelniania i kryptografii (np. ważności certyfikatów). Zwykle host synchronizuje czas z serwerem czasu (lokalnym lub publicznym), a serwery w organizacji mogą tworzyć własną hierarchię źródeł czasu.

Podsumowanie

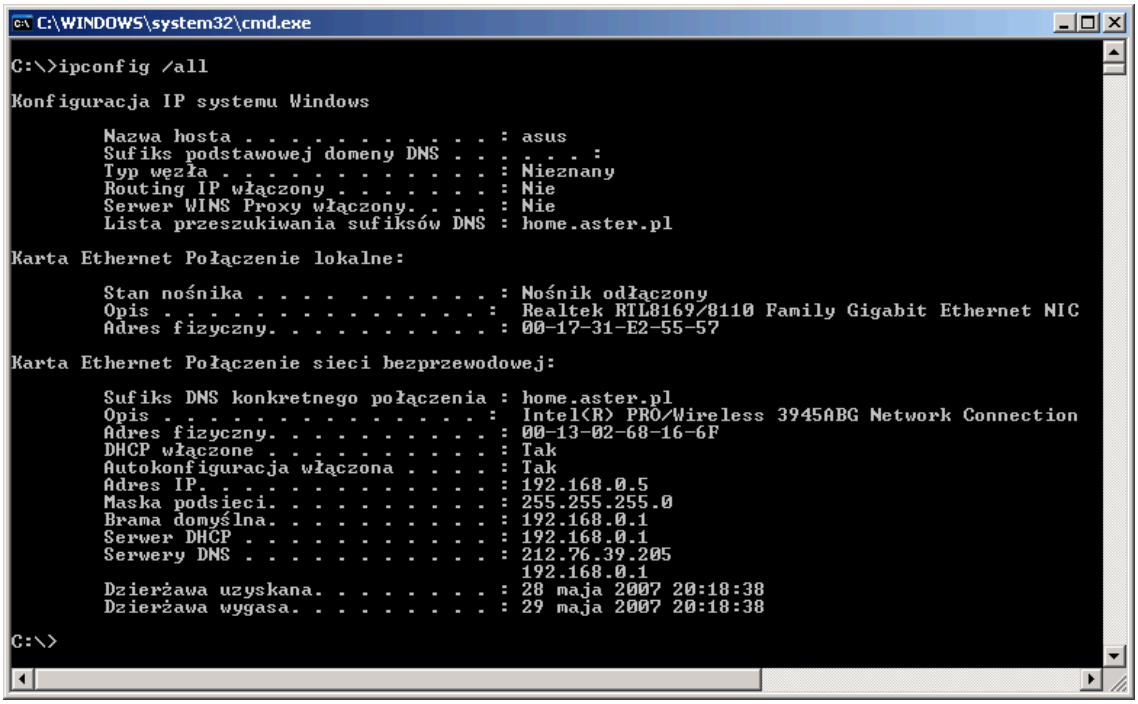
Zamieszczona lista nie jest pełnym przeglądem protokołów sieciowych – jest ich znacznie więcej. Istnieją protokoły zdalnej pracy z GUI (RDP, VNC), protokoły sieciowych systemów plików, protokoły komunikacji aplikacji i chmur, protokoły strumieniowania multimediów, systemy kolejkowe i wiele innych. W kolejnych rozdziałach skupimy się na tych elementach stosu TCP/IP, które są niezbędne do zrozumienia adresowania, routingu oraz działania protokołów transportowych i usług na nich opartych.

1.5. Diagnostyka sieci TCP/IP

Zarówno systemy Windows, jak i Unix/Linux oferują podstawowe narzędzia do diagnostyki sieci: sprawdzania konfiguracji interfejsów, testowania łączności, śledzenia trasy pakietów oraz weryfikacji działania usług takich jak DNS. Poniżej zebrano te polecenia, które najczęściej wykorzystuje się w praktyce.

Konfiguracja IP

Do wyświetlania bieżącej konfiguracji TCP/IP w systemie Windows służy polecenie `ipconfig` (najczęściej z opcją `/all`). W systemach Unix/Linux historycznie używano polecenia `ifconfig`, jednak współcześnie częściej spotkacie zestaw narzędzi `ip`, np. `ip addr` (adresy) oraz `ip link` (interfejsy).



Wyświetlenie przykładowej konfiguracji IP w systemie Windows.

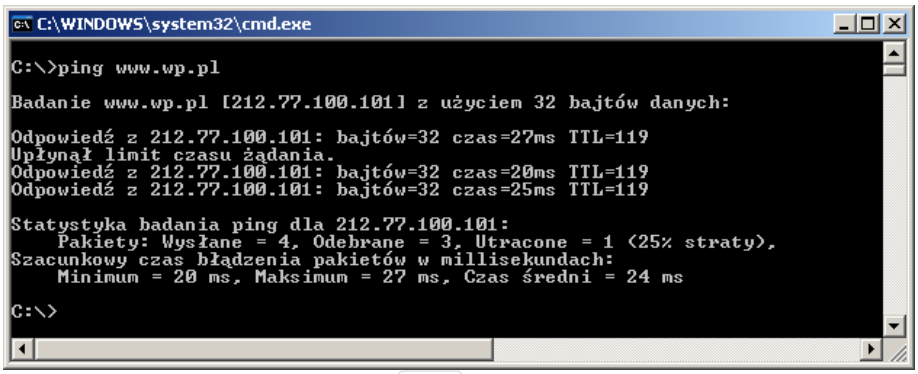
Po wykonaniu polecenia `ipconfig /all` system wyświetla informacje ogólne oraz szczegóły dla każdego z interfejsów sieciowych, w tym:

- stan interfejsu (aktywny/nieaktywny);
- informacje o karcie sieciowej, opis oraz adres MAC (warstwa 2 modelu OSI);
- informację, czy konfiguracja jest statyczna, czy pobierana z DHCP;
- adres IP interfejsu sieciowego;
- maskę sieci/podsieci;
- domyślną bramę (tam trafią pakiety kierowane poza sieć lokalną);
- adres serwera DHCP (tylko w przypadku konfiguracji dynamicznej);
- adresy serwerów DNS;
- informacje o dzierżawie adresu (czas obowiązywania i odnowienie) – tylko dla DHCP.

Diagnostyka łączności

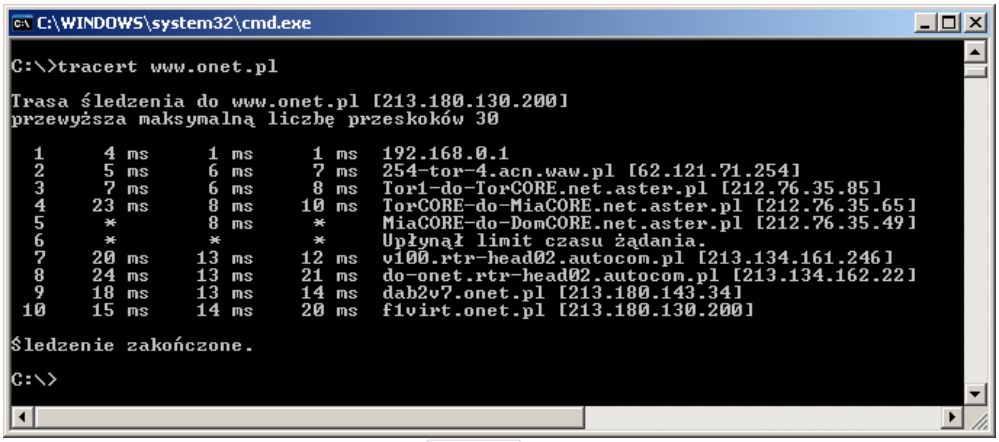
Podstawowym narzędziem do diagnostyki sieci jest program `ping`. Wykorzystuje on protokół ICMP i wysyła do adresu docelowego zapytania, mierząc czas odpowiedzi oraz liczbę utraconych pakietów. Jeżeli host docelowy działa i odpowiada na ICMP, zobaczycie komunikaty w rodzaju `Odpowiedź z ...` oraz podsumowanie statystyk po zakończeniu.

Trzeba jednak pamiętać, że w wielu sieciach ruch ICMP bywa blokowany przez zapory (firewall) lub ograniczany polityką bezpieczeństwa. W takim przypadku brak odpowiedzi z `ping` nie musi oznaczać awarii hosta – może oznaczać jedynie, że host (albo urządzenie po drodze) nie odpowiada na ICMP.



Przykładowy wynik wykonania polecenia `ping`.

Drugim bardzo ważnym narzędziem jest śledzenie trasy pakietów: w Windows używa się `tracert`, a w Unix/Linux `traceroute`. Narzędzia te pokazują kolejne „skoki” (routery) na drodze do celu oraz orientacyjne czasy odpowiedzi. Działają one na zasadzie wysyłania pakietów z rosnącą wartością TTL i obserwowania komunikatów ICMP zwracanych przez routery po drodze.



Wynik przykładowego wykonania programu `tracert` w systemie Windows.

Podobnie jak w przypadku `ping`, wyniki `traceroute` mogą być niepełne, jeśli urządzenia po drodze filtrują ICMP lub ograniczają odpowiedzi. Mimo tego jest to bardzo przydatne narzędzie do wykrywania problemów z trasą lub zauważania nietypowych „objazdów” w routingu.

W Internecie dostępne są narzędzia do graficznej prezentacji trasy. Przykładem jest witryna geotraceroute.com. Takich narzędzi jest więcej – wystarczy wyszukać hasło *visual traceroute*.

Diagnostyka DNS i połączeń

Jeśli podejrzewacie problem z tłumaczeniem nazw, warto użyć narzędzi DNS. W Windows często spotyka się `nslookup`, a w Unix/Linux (oraz na wielu systemach administratorskich) standardem jest `dig`. Pozwalają one sprawdzić, czy i jak serwer DNS tłumaczy nazwę na adres IP, jakich rekordów użyto oraz czy odpowiedź pochodzi z cache.

Do sprawdzania aktywnych połączeń, portów nasłuchujących i stanu gniazd sieciowych wykorzystuje się polecenia `netstat` (starsze, ale nadal popularne) oraz `ss` (współcześnie często preferowane w Linux). Tego typu narzędzia pomagają odpowiedzieć na pytania: „czy usługa działa i nasłuchuje?”, „czy połączenie TCP zostało zestawione?”, „na jakich portach system ma otwarte gniazda?”.