

Systemy operacyjne

Marek Grochowski

Piotr Ablewski

Jacek Kobus

Wydział Fizyki, Astronomii i Informatyki Stosowanej UMK (2024/25)

`grochu@is.umk.pl`

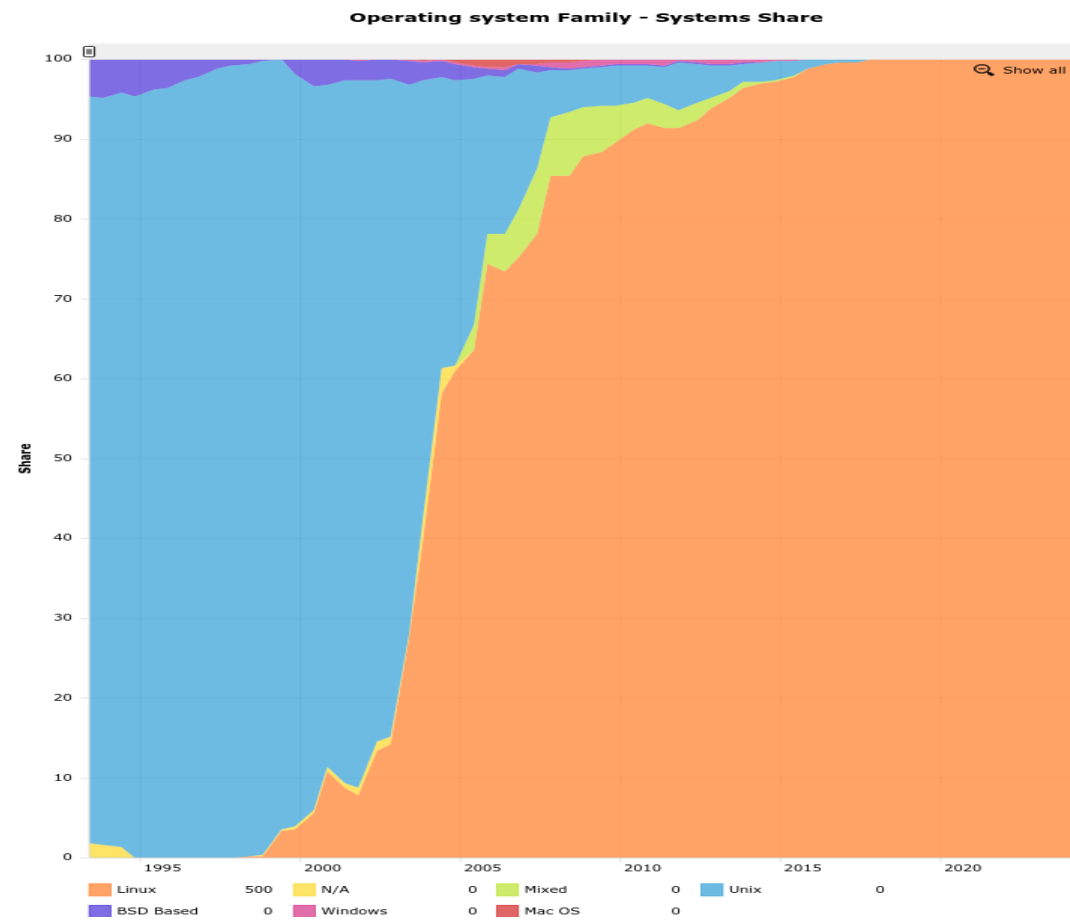
`https://www.fizyka.umk.pl/~grochu/so/`

ostatnia aktualizacja: 26 listopada 2024

TOP500: 5 najszybszych superkomputerów (6/2024)

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	Frontier - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,699,904	1,206.00	1,714.81	22,786
2	Aurora - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel DOE/SC/Argonne National Laboratory United States	9,264,128	1,012.00	1,980.01	38,698
3	Eagle - Microsoft NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR, Microsoft Azure Microsoft Azure United States	2,073,600	561.20	846.84	
4	Supercomputer Fugaku - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21	29,899
5	LUMI - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	2,752,704	379.70	531.51	7,107

TOP 500: udział systemów operacyjnych (6-2024) ¹



¹<https://www.top500.org/statistics/overtime/>

Windows kontra Linux wg E.S.Raymonda²

ESR na swoim blogu zatytułowanym *Last phase of the desktop wars?* napisał m.in:

- The two most intriguing developments in the recent evolution of the Microsoft Windows operating system are Windows System for Linux (WSL) and the porting of their Microsoft Edge browser to Ubuntu.

[...] WSL allows unmodified Linux binaries to run under Windows 10. No emulation, no shim layer, they just load and go.

- Microsoft developers are now landing features in the Linux kernel to improve WSL. [...] To understand why, we need to notice how Microsoft's revenue stream has changed since the launch of its cloud service in 2010.
- Ten years later, Azure makes Microsoft most of its money. The Windows monopoly has become a sideshow, with sales of conventional desktop PCs (the only market it dominates) declining.
- Proton is the emulation layer that allows Windows games distributed on Steam to run over Linux.

... the end state this all points at is: New Windows is mostly a Linux kernel, there's an old-Windows emulation over it, but Edge and the rest of the Windows user-land utilities don't use the emulation. The emulation layer is there for games and other legacy third-party software.

²25/09/2020: <https://esr.ibiblio.org/?p=8764>

Program wykładu

1. Wprowadzenie

- Co to jest system komputerowy?
- Co to jest system operacyjny?
- Historia komputerów i systemów operacyjnych

2. Architektura współczesnych komputerów

- Architektura i działanie procesora. Przetwarzanie rozkazów
- Rodzaje i hierarchia pamięci
- Magistrale i urządzenia peryferyjne

3. Jak system komputerowy wykonuje programy?

- Monitor prosty, buforowanie, spooling
- Wieloprogramowość
- Systemy z podziałem czasu
- Systemy rozproszone, systemy czasu rzeczywistego

4. Przerwania

- Systemy z obsługą przerwań
- Struktura wejścia-wyjścia
- Dualny tryb pracy
- Funkcje systemowe

5. Procesy

- Model procesu i jego realizacja
- Stany procesów, zarządzanie procesami
- Wątki, wątki jądra, procesy lekkie
- Zadania w Linux
- Wywłaszczenie

6. Zarządzanie procesami

- Algorytmy przydziału procesora
- Priorytety
- Planista CFS
- Komunikacja międzyprocesowa, sygnały

7. Synchronizowanie procesów. Zakleszczenia.

- Obszary krytyczne i wyścigi
- Wzajemne wyłączenie z aktywnym czekaniem
- Problem producenta konsumenta
- Semafor
- Monitory
- Zakleszczenia

8. Zarządzanie pamięcią

- Zarządzanie pamięcią bez wymiany i stronicowania
- Wymiana
- Pamięć wirtualna
- Segmentacja

9. Zarządzanie przestrzenią dyskową

- Rodzaje plików
- Partycje i systemy plików, systemy plików z kroniką
- Zarządzanie logicznymi wolumenami
- Macierze dyskowe

10. Struktura systemów operacyjnych

- Systemy monolityczne
- Systemy warstwowe
- Maszyny wirtualne
- Model klient-serwer

11. Przykłady systemów operacyjnych

- MSDOS, Windows 95/98
- Windows 2000/NT/...
- Unix, GNU/Linux
- ...

System komputerowy

- użytkownicy (ludzie, maszyny, inne komputery)
- programy użytkowe (kompilatory, edytory, systemy baz danych, gry)
- system operacyjny
- sprzęt (procesor, pamięć, urządzenia wejścia–wyjścia)

System komputerowy wg Tanenbauma

systemy bankowe	rezerwacja biletów	gry
kompilatory	edytory	powłoki
system operacyjny		
język maszynowy		
mikroprogramowanie		
urządzenia fizyczne		

System komputerowy

Poziomy abstrakcji współczesnych systemów komputerowych
(wg Null i Lobur)

6	użytkownik	programy wykonywalne
5	języki wysokiego poziomu	C, C++, Java, FORTRAN, ...
4	assembler	kod assemblera
3	oprogramowanie systemowe	system operacyjny, biblioteki
2	maszyna	architektura zbioru rozkazów
1	sterowanie	mikrokod lub skonfigurowane sprzętowo
0	logika cyfrowa	obwody, bramki, itd.

Co to jest system operacyjny?

- System operacyjny jest **programem**; działa jako pośrednik między użytkownikiem komputera a sprzętem komputerowym; tworzy środowisko, w którym użytkownik może wykonywać programy.
- System operacyjny **nadzoruje i koordynuje** posługiwanie się sprzętem przez programy użytkowe, które pracują na zlecenie użytkowników.
- System operacyjny jest odpowiedzialny za
 - zarządzanie zasobami komputera
 - tworzenie wirtualnej maszyny (warstwy abstrakcji) dla programów użytkowych i programistów
- **Jądro (*kernel*)** – część systemu operacyjnego, która działa nieustannie; wszystkie pozostałe programy są programami użytkowymi.

Cechy dobrego systemu operacyjnego

- funkcjonalność
- wydajność (procesor powinien pracować dla użytkownika)
- skalowalność (zwiększenie obciążenia i rozbudowa sprzętu)
- niezawodność (dostępność, *five nines*)
- łatwość korzystania i zarządzania

Prehistoria rozwoju komputerów

- Abak, liczydło (znane od około 3000 lat)
- wprowadzenie cyfr arabskich: Gerbert z Aurillac (koniec X w.), Leonardo Fibonacci (przełom XII i XIII w.)
- pałeczki Nepera (wykorzystanie logarytmów):
John Neper (Napier), Henry Briggs (początek XVII w.)
- zegar liczący Wilhelma Schickarda:
pierwsza maszyna licząca (około 1623)
- Pascaline: arytmometr Blaise'a Pascala (1642)
- programowalne urządzenie włókiennicze: Jaques de Vaucanson (1745)
- pierwsze uniwersalne krosno wykorzystujące karty perforowane:
Joseph-Marie Jacquard (pierwsza połowa XIX w.)

Prehistoria rozwoju komputerów

- Maszyna różnicowa Charlesa Babbage'a (około 1823).
Każdy wielomian może być wyliczany w oparciu o różnice skończone:

$$f_n = n^2 + n + 41$$

$$\Delta f_n = f_n - f_{n-1} = 2n$$

$$\Delta^2 f_n = \Delta f_n - \Delta f_{n-1} = 2$$

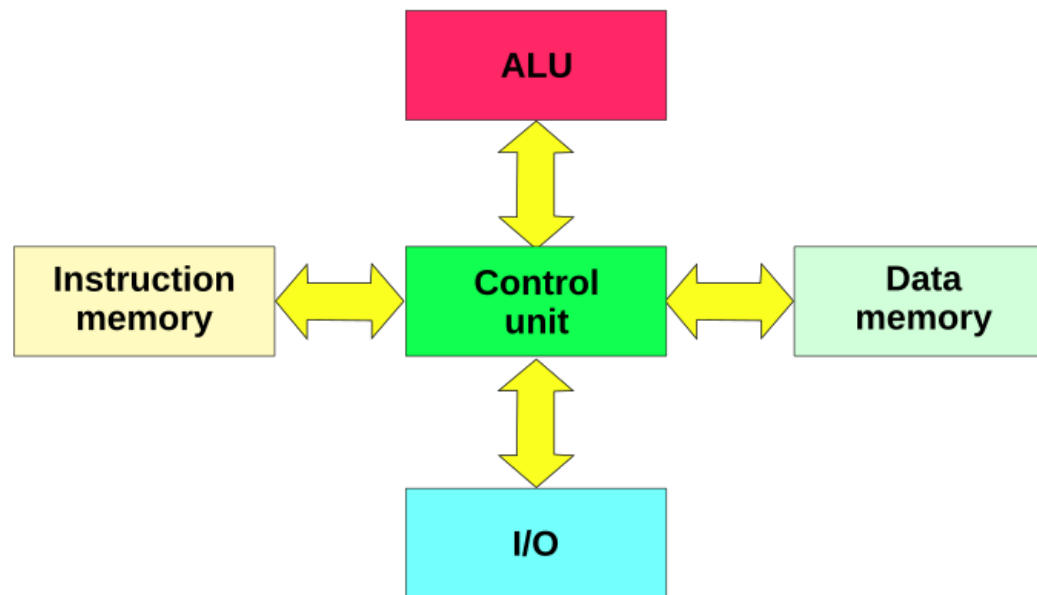
$$f_n = f_{n-1} + \Delta f_n = f_{n-1} + \Delta f_{n-1} + \Delta^2 f_n$$

n	f_n	Δf_n	$\Delta^2 f_n$	$f_{n-1} + \Delta f_{n-1} + \Delta^2 f_n$
0	41	0	2	43
1	43	2	2	47
2	47	4	2	53
3	53	6	2	61
4	61	8	2	71
5	71	10	2	83

Prehistoria rozwoju komputerów

- maszyna różnicowa braci Scheutz (1855): szybkość 33-44 32-cyfrowe liczby na minutę
- maszyna analityczna Babbage'a (1835; Luis Manabrea, Ada Byron)
- *Prawa myślenia* (algebra Boole'a): George Boole (1854)
- karty perforowane jako nośniki informacji: Herman Hollerith (przełom XIX i XX w.)
- pierwszy kalkulator elektroniczny (1939): John Atanasoff, Clifford Berry
- Automatic Sequence Controlled Calculator (ASCC, Harvard Mark I, kalkulator automatycznego sterowania sekwencyjnego): Howard Aiken + inżynierowie IBM (1944) (dodawanie – 0.3 sek, mnożenie – 6 sek)

Architektura Harwardzka



- ASCC, Harvard Mark I
- osobna pamięć i magistrale dla danych i instrukcji

Generacje komputerów

generacja	lata	technologia	szybkość (operacji/sek)
1	1946-1957	lampa próżniowa	40 000
2	1958-1964	tranzystor	200 000
3	1965-1971	mała i średnia skala scalenia	1 000 000
4	1972-1977	duża skala scalenia	10 000 000
5	1978-	bardzo duża skala scalenia	> 100 000 000

SSI (*Small Scale Integration*) mała skala integracji, 10-100 elem./układ

MSI (*Medium Scale Integration*) średnia skala integracji, 100-1000 elem./układ

LSI (*Large Scale Integration*) duża skala integracji, 1000-10000 elem./układ

VLSI (*Vary Large Scale Integration*) bardzo duża skala integracji, powyżej 10000 elem./układ

Historia rozwoju komputerów i systemów operacyjnych

- 1642-1945 (0. generacja): mechaniczne maszyny liczące
- 1945-1953 (1. generacja): komputery na lampach próżniowych
 - brak systemów operacyjnych i języków programowania
 - bezpośrednie programowanie w języku maszynowym
 - przeprowadzanie wyłącznie obliczeń numerycznych
 - zastosowanie kart dziurkowanych do wprowadzania danych (początek lat 1950.)

ENIAC *Electronic Numerical Integrator and Computer*

Elektroniczny integrator numeryczny i komputer (1946-1955, J.Mauchly, J.P.Eckert):

- 18 tys. lamp próżniowych, 70 tys. oporników, 10 tys. kondensatorów, 1.5 tys. przekaźników, 6 tys. ręcznych przełączników
- 30 ton, 170 m², moc 160kW
- 5000 operacji dodawania na sekundę
- maszyna dziesiętna (każda liczba była reprezentowana przez pierścień złożony z 10 lamp)
- ręczne programowanie przez ustawianie przełączników i wtykanie kabli

Park w Bletchley, Anglia, 1939-1945

Government Code and Cipher School, siedziba wywiadu, gdzie łamano niemieckie szyfry przy pomocy ulepszonej „bomby Rejewskiego” (M.Rejewski, J.Różycki, H.Zygalski³), a następnie maszyn Heath Robinson, Mark 1 Colossus (1943) i Mark 2 Colossus (1944).

Mark 2 Colossus:

- 2400 lamp elektronowych
- 5 czytników taśm (5×5 tys. znaków na sekundę, bufony)
- arytmetyka binarna
- testowanie Boole’owskich operacji logicznych
- rejestry pamięci elektronicznej sterowane automatycznie
- realizacja podprogramów dla wykonywania określonych funkcji
- wyniki wyprowadzane za pomocą elektrycznej maszyny do pisania

³Zob. [Otwarcie Centrum Szyfrów Enigma, Centrum Szyfrów w Poznaniu](#)

EDVAC *Electronic Discrete Variable Computer*

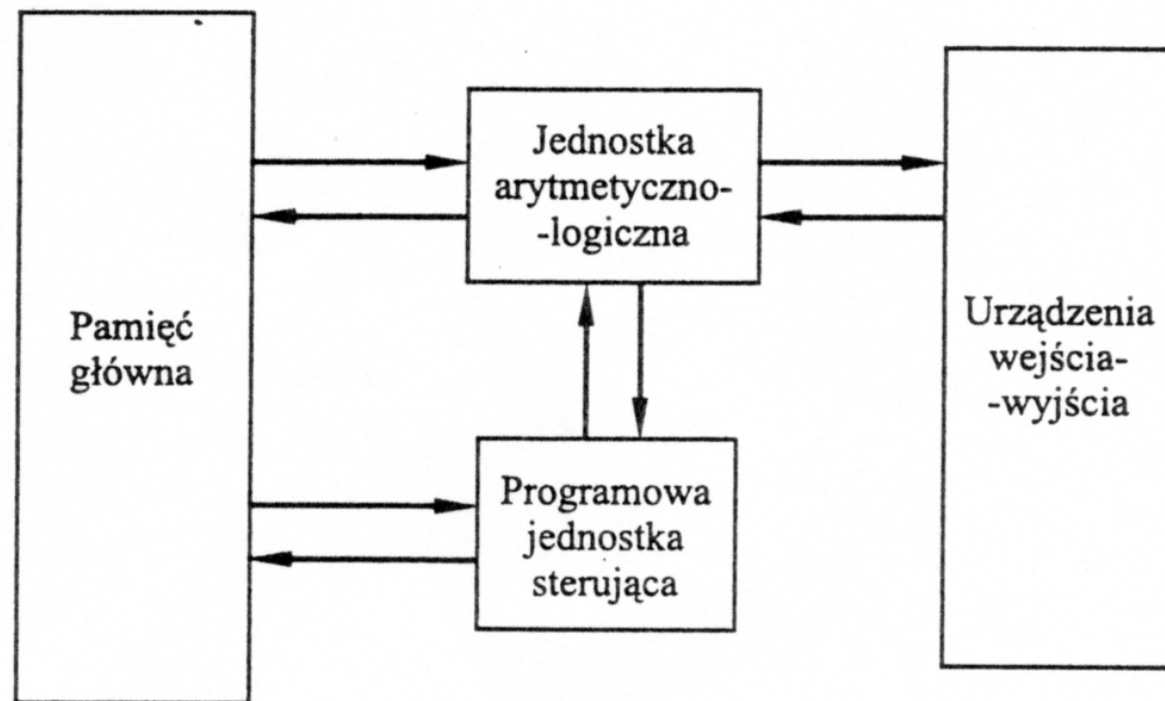
Komputer wg Johna von Neumanna (1946, IAS⁴)
(pomysł Mauchly'ego i Eckerta):

- pamięć główna przechowująca dane i rozkazy,
1000 słów 40 bitowych;
2 instrukcje (20-to bitowe) w słowie
każdy element pamięci ma unikalny identyfikator - adres
- jednostka arytmetyczno-logiczna (ALU, *Arithmetic-Logic Unit*)
wykonująca działania na danych binarnych
- jednostka sterująca, która interpretuje rozkazy z pamięci i powoduje ich wykonanie
- urządzenia wejścia-wyjścia, których pracą kieruje jednostka centralna

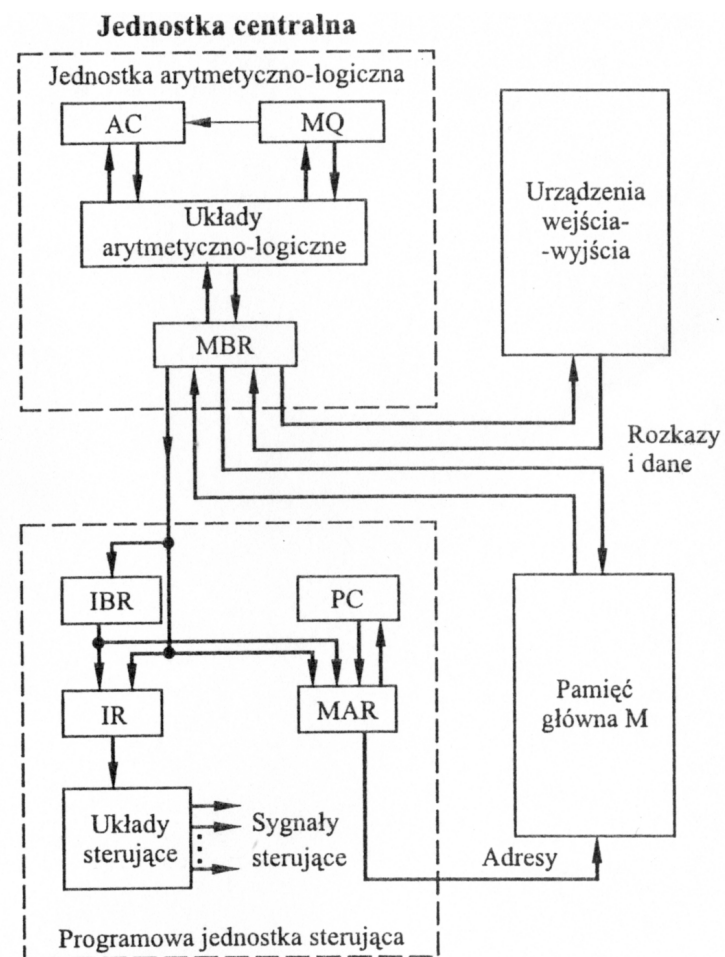
Maszyny von Neumanna – komputery o takiej ogólnej strukturze

⁴Institute for Advanced Study, Princeton

Maszyna von Neumanna



RYSUNEK 2.1. Struktura komputera IAS



RYSUNEK 2.3. Struktura komputera IAS

Jednostka sterująca uruchamia IAS, pobierając rozkaz z pamięci i wykonując go (jeden rozkaz w określonym momencie)

- rejestr buforowy pamięci (MBR – memory buffer register) – przechowywanie słowa, które ma być pobrane lub odesłane do pamięci
- rejestr adresowy pamięci (MAR – memory address register) – adres w pamięci słowa, które ma być zapisane/odczytane z MBR
- rejestr rozkazów (IR – instruction register) – 8-bitowy kod operacyjny rozkazu, który jest wykonywany
- buforowy rejestr rozkazów (IBR – instruction buffer register) czasowo przechowuje podręczny rozkaz ze słowa w pamięci
- licznik programu (PC – program counter) – adres kolejnej pary rozkazów
- akumulator (AC – accumulator) oraz rejestr mnożenia i dzielenia (MQ – multiplier-quotient) – czasowe przechowywanie argumentów i wyników operacji prowadzonych przez ALU.

Cykle komputera IAS:

- **cykl rozkazu:** kod operacji następnego rozkazu jest ładowany do IBR, a część adresowa – do MAR
- **cykl wykonania:** układy sterujące rozpoznają kod operacji i wykonują rozkaz, wysyłając odpowiednie sygnały sterujące, które powodują, że przenoszone są dane lub ALU wykonuje operację

Komputer IAS ma 21 rozkazów:

- przenoszenie danych
- rozgałęzienie bezwarunkowe
- rozgałęzienie warunkowe
- arytmetyka
- modyfikowanie adresu

Wykonanie operacji mnożenia wymaga wykonania 39 podoperacji.

Lista rozkazów IAS

Kod operacji	Reprezentacja symboliczna	Opis
00001010	LOAD MQ	przenieś zawartość MQ do AC
00001001	LOAD MQ,M(X)	przenieś zawartość komórki X do MQ
00000001	LOAD M(X)	przenieś M(X) do akumulatora
00001101	JUMP M(X,0:19)	pobierz następny rozkaz z lewej połowy M(X)
00001110	JUMP M(X,20:39)	pobierz następny rozkaz z prawej połowy M(X)
00001111	JUMP+M(X,0:19)	jeśli liczba w AC ≥ 0 pobierz nast. rozkaz z lewej połowy M(X)
00010000	JUMP+M(X,20:39)	jeśli liczba w AC ≥ 0 pobierz nast. rozkaz z prawej połowy M(X)
00000101	ADD M(X)	dodaj M(X) do AC i wynik umieść w AC
00000110	SUB M(X)	odejmij M(X) od AC i wynik umieść w AC
00001011	MUL M(X)	pomnóż M(X) przez MQ i umieść najbardziej znaczące bity w AC, a najmniej w MQ
00001100	DIV M(X)	podziel zawartość AC przez M(X), umieść iloraz w MQ, resztę w AC
00010101	LSH	pomnóż AC przez 2 (przesuń w lewo o jedną pozycję)
00010101	RSH	podziel AC przez 2 (przesuń w prawo o jedną pozycję)
00010010	STOR M(X,8:19)	zamień lewe pole adresowe M(X) na 12 bitów AC z prawej

Pierwsze komputery komercyjne

W roku 1947 Mauchly i Eckert tworzą Eckert-Mauchly Computer Corporation. Powstaje pierwszy komputer komercyjny UNIVAC I (*Universal Automatic Computer*), który może realizować

- macierzowe rachunki algebraiczne
- problemy statystyczne
- obliczanie premii dla firm ubezpieczeniowych

UNIVAC I został wykorzystany do powszechnego spisu w 1950 r.

IBM wyprodukował swój pierwszy komputer elektroniczny z przechowywanym programem do zastosowań naukowych w 1953 r. (model 701). Model 702 (1955) znalazł zastosowanie w biznesie.

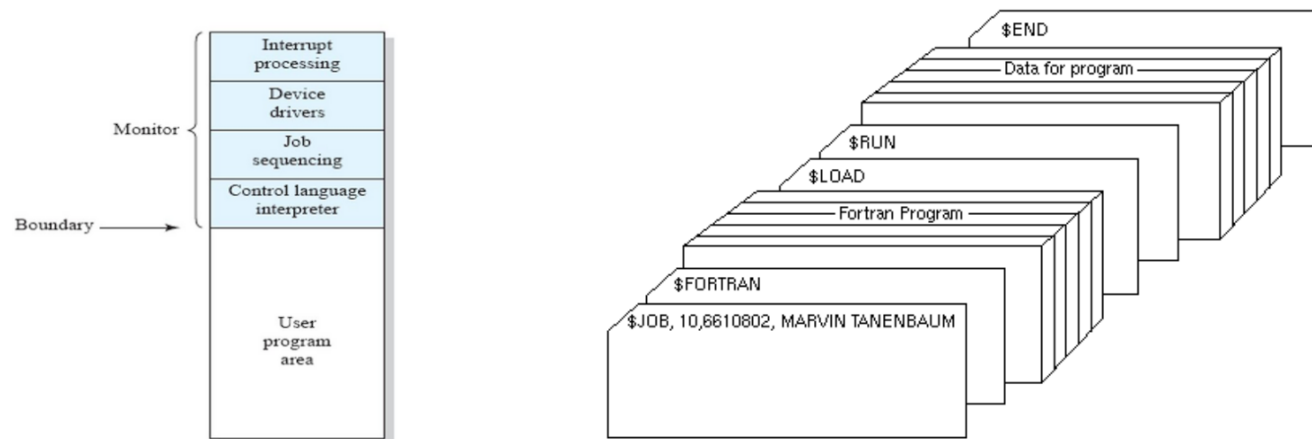
2. generacja (1953-1964): komputery tranzystorowe

- zwiększona niezawodność
- rozgraniczenie roli operatora i programisty
- system wsadowy i praca pośrednia (wczytywanie i drukowanie *off-line*)
- proste monitory i karty sterowania zadaniami (*Fortran Monitor System*, *Job Control Language*)

Prosty **monitor** wraz z językiem kontroli zadań (JCL) umożliwiał przetwarzanie wsadowe (*batch processing*).

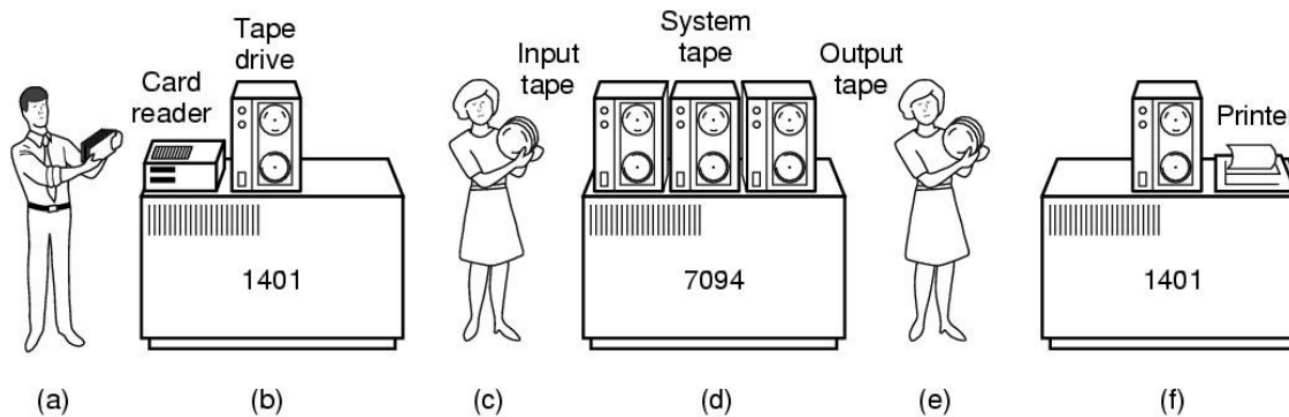
Bardziej złożone jednostki ALU oraz sterujące, ulepszone języki programowania, rozpoczęto dostarczanie **oprogramowania systemowego**.

Monitor rezydujący w pamięci



- specjalne karty sterujące, instrukcje monitora
- monitor i program rezydują jednocześnie w wydzielonych obszarach pamięci
- użytkownik nie ma bezpośredniego dostępu do procesora
- czytniki kart i drukarki powolne względem jednostki obliczeniowej

Wczesne systemy przetwarzania wsadowego



- praca w trybie pośrednim (off-line) - przyspieszenie obliczeń poprzez ładowanie zadań do pamięci z taśm oraz odczyt kart i drukowanie wierszy przy użyciu mniejszych maszyn
- nie trzeba wprowadzać żadnych zmian w programach aplikacji, aby przejść z trybu bezpośredniego na tryb off-line
- możliwość korzystania z wielu systemów wejściowych i wyjściowych

Rozwój komputerów IBM z serii 700

- zwiększenie pamięci
- skrócenie czasu cyklu pamięci
- wprowadzenie kanałów danych
- zastosowanie multipleksera

Kanał danych: niezależny moduł wejścia-wyjścia z własnym procesorem i własną listą rozkazów.

Multiplekser: szereguje dostęp procesora i kanałów danych do pamięci, urządzenia mogą działać niezależnie.

3. generacja: komputery zbudowane z układów scalonych

- ujednolicenie linii produkcyjnych;
małe i duże komputery z ujednoliconym systemem operacyjnym
(ta sama lista rozkazów, ale różna wielkość pamięci, szybkość procesorów, wydajność)
- wieloprogramowość, spooling
- systemy z podziałem czasu:
 - interaktywny dostęp wielu użytkowników, współbieżne działanie wielu programów, ochrona pamięci, systemy plików,
 - *Compatible Time-Sharing System* (CTSS)
 - *Multiplexed Information and Computing Service* (MULTICS)
 - duży wpływ na Unix, Linux, iOS, Android
- minikomputery (DEC PDP)

System IBM 360 (1964):

- rodzina komputerów o ujednoliconej architekturze do ogólnych zastosowań (naukowych, biznesowych)
- podobna/identyczna liczba rozkazów
- podobny/identyczny system operacyjny (OS/360), przenośny (w teorii)
- cena uzależniona od wydajności: moc obliczeniowa, liczba urządzeń wej-wy, rozmiar pamięci
- upowszechnił wieloprogramowość,
- spooling wyeliminował małe komputery do obsługi kart
- 8-mio bitowy bajt, 32 bitowe słowo, przerwania, pamięć wirtualna (w modelu 67)
- nadal długi czas przetwarzania wsadowego, brak interaktywności

DEC PDP-8 (1964) – pierwszy minikomputer (magistrala Omnibus)

4. generacja (1980-): komputery z układami VLSI

- układy o dużym stopniu scalenia (VLSI)
- powstanie stacji roboczych i komputerów osobistych (architektura minikomputerów)
- sieciowe systemy operacyjne
- rozproszone systemy operacyjne (układy masywnie równoległe)
- systemy operacyjne urządzeń mobilnych
- wbudowane systemy operacyjne

Zalety układów scalonych:

- koszt mikroukładu niezmienny pomimo wzrostu gęstości upakowania
- gęstsze upakowanie to większa szybkość działania układu
- zmniejszenie wymiarów komputera
- mniejsze zapotrzebowanie na moc i łatwiejsze chłodzenie
- połączenia wewnątrz układu scalonego są bardziej niezawodne, niż połączenia lutowane

Początki komputeryzacji w Polsce

- Państwowy Instytut Matematyczny, Grupa Aparatów Matematycznych kierowana przez dra Henryka Greniewskiego:
 - Analogowy Analizator Równań Algebraicznych Liniowych (ARAL, 1954, K.Bochenek): 400 lamp elektronowych, rozwiązywanie układu co najwyżej 8 (niekoniecznie) liniowych równań różniczkowych.
 - Analogowy Analizator Równań Różniczkowych (ARR, L.Łukaszewicz); Elektroniczna Maszyna Automatycznie Licząca (EMAL, R.Marczyński).
 - XYZ (1958) wzorowany na IBM 701, pamięć naddźwiękową, 800 operacji/sek., przerzutniki z maszyny sowieckiej BESM 6; ZAM 2 (ulepszona wersja XYZ).
- opracowanie Systemu Automatycznego Kodowania (SAKO, 1960), Polski FORTRAN.
- Odra 1002 (1962) – pierwszy komputer z zakładów Elwro, Odra 1003 (1964) – pierwszy seryjnie produkowany komputer (ostatni wyłączono 30/04/2010 po 34 latach pracy)
- 16-bitowy minikomputer K-202 J.Karpińskiego (1970-1973), pierwszy polski komputer zbudowany z użyciem układów scalonych, przewyższał pod względem szybkości pierwsze IBM PC oraz umożliwiał wielozadaniowość, wielodostępność i wieloprocesorowość.
- minikomputer Mera (1978)⁵

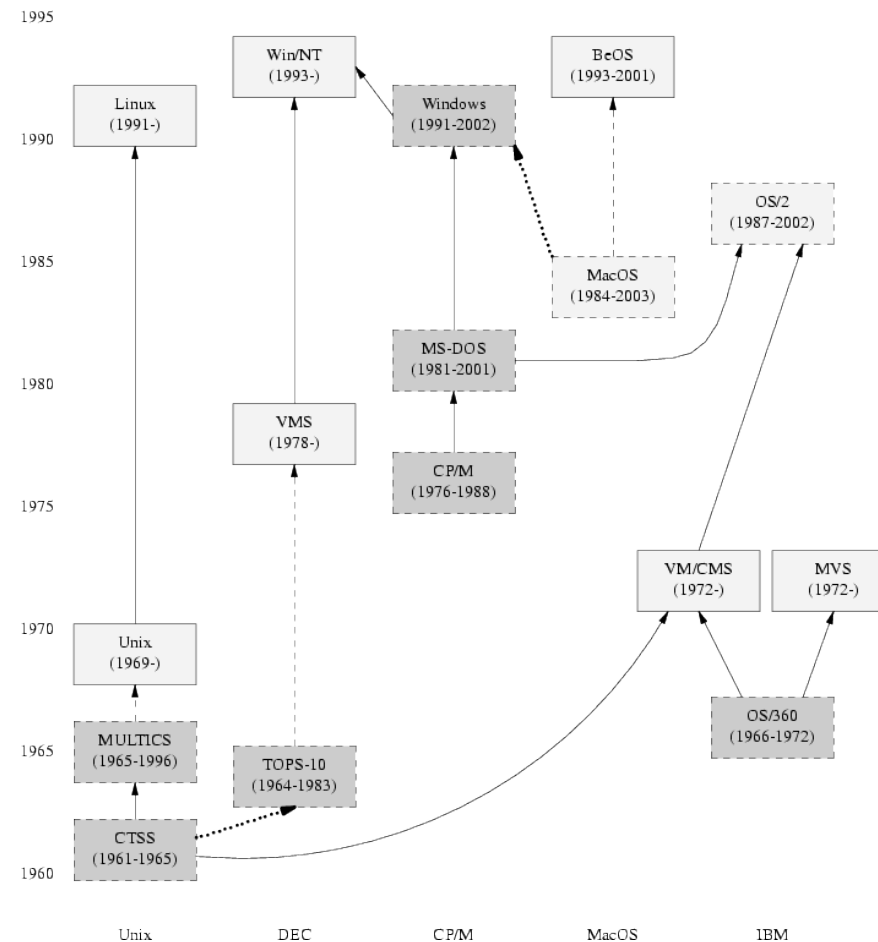
⁵Zob. System Mera 400

Systemy operacyjne z początków komputeryzacji w Polsce

- SOK-1 - tylko jeden program (komputer K-202)
- SODA System Operacyjny Dwu Aktywny (Odra), dwa programy
- Egzekutor RTX (*Real Time Executive*), Mera 300
- Ładowacz (Mera 305)
- MSO 300 (Mera 306), system ogólnego przeznaczenia
- SOWA System Operacyjny Wielo Aktywny - potem nazwany CROOK
- CROOK 1-5, komputery K-202 i Mera 400, uniwersalny, wielozadaniowy, wielodostępowy system operacyjny z hierarchicznym systemem plików⁶

⁶Zob. <https://mera400.pl/CROOK>

Historia systemów operacyjnych wg E.S.Raymonda⁷

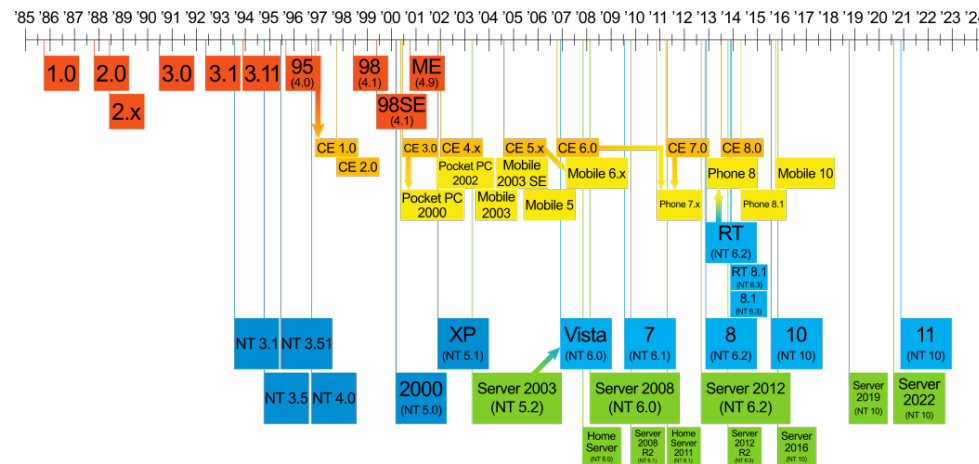


⁷ *UNIX Sztuka programowania*, Wydawnictwo Helion, Gliwice, 2004. Także http://en.wikipedia.org/wiki/Timeline_of_operating_systems

MS-DOS, Windows

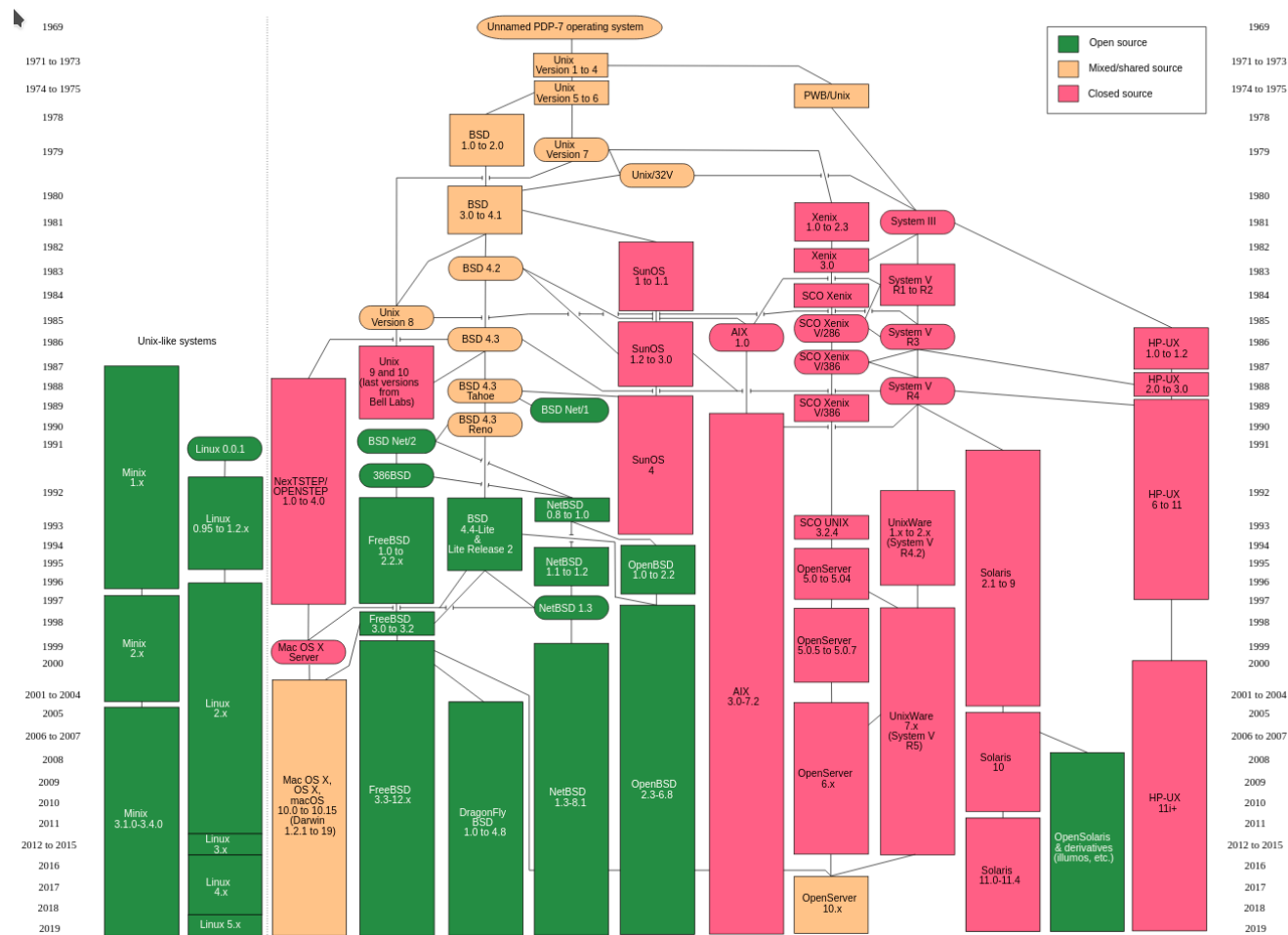
- 1974 pierwszy 8-mio bitowy procesor 8080 ogólnego przeznaczenia,
- dyskowy system CP/M (*Control Program for Microcomputer*), Gary Kildall, rozwijany przez Digital Research zdominował rynek mikrokomputerów (m.in. Atari, ZX Spectrum)
- 1981 IBM PC, poszukiwany system, Bill Gates doradza CP/M ale nie dochodzi do porozumienia z Digital Research
- Bill Gates odkupuje DOS od lokalnej firmy i proponuje MS-DOS, system jest dołączany do sprzętu
- 1983 IBM PC/AT silna pozycja MS-DOS
- 1984 Steve Jobs dostrzega potencjał GUI, komputer Lisa (porażka), potem Apple Macintosh - wielki sukces, Mac OS X (obecnie macOS) adoptuje elementy jądra UNIX (BSD)
- 1985 Windows 1.0 - początkowo graficzna nakładka na MS-DOS

- 1995 Windows 95 pierwszy samodzielny system (MS-DOS do rozruchu i zgodności oprogramowania)
- Windows NT (*New Technology*) napisany na nowo, 32 bitowy, zgodny z W95/W98
- od 2000 dwie linie:⁸
 - serwerowa: Windows Server 2003, Windows Server 2008, ...
 - kliencka: oparte o Windows XP, Windows 7, 8, 10, ...



⁸https://pl.wikipedia.org/wiki/Microsoft_Windows

Historia systemu operacyjnego Unix



Historia Unix

- 1969 port gry Space Travel na PDP-7 (Ken Thompson), powstaje system bazujący na MULTICS dla jednego użytkownika
- 1969 Unix, Ken Thompson, Denis Ritchie (AT&T Bell Labs)
- 1971 język C, D. Ritchie + kompilator
- 1974 udostępnienie źródeł Unix-a (przepisanych z użyciem C)
- 1977 Unix BSD rozwijany na Uniwersytecie Kalifornijskim w Berkeley (K.Thompson i B.Joy) - wiele ważnych zmian (pam. wirt, TCP/IP)
- 1978 zaczyna działać Santa Cruz Operations (Xenix, SCO Unix)
Microsoft uzyskuje licencję na system Unix (Xenix)

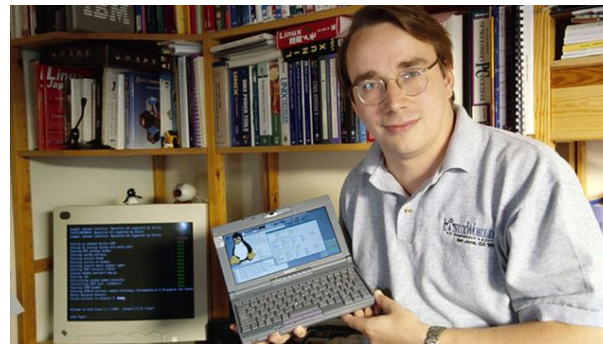


Historia Unix

- 1983 firma AT&T komercjalizuje Unix System V
- brak standaryzacji
 - IBM – AIX (SVR3)
 - DEC – Ultrix (4.2 BSD)
 - SUN – Solaris (SVR4)
 - HP – HP-UX (SVR4)
 - SGI – IRIX (BSD)
 - Microsoft i SCO – Xenix (Unix Version 7)
- 1988 POSIX, Portable Operating System Interface, standard definiujący interfejsy systemowe i programowania aplikacji (API) na poziomie użytkownika, w celu zapewnienia zgodności oprogramowania (przenośności) z wariantami Uniksa i innych systemów operacyjnych

Historia GNU Linux

- 1984 GNU (*Gnu's Not Unix*), wolny (o swobodnym dostępie), uniksopodobny system operacyjny⁹
- 1985 GNU Manifesto, R. Stallman, powstaje Free Software Foundation
- 1989 oprogramowanie GNU udostępnione na licencji GPL (copyleft)
- 1991 jądro Linux udostępnione na licencji GNU, Linus Torvalds
jądro wykorzystane w GNU/Linux
- 2000 powstaje The Linux Foundation



⁹ www.gnu.org

GNU/Linux

- GNU – Gnu's Not Unix, wolny, kompatybilny z Unix system operacyjny
- Linux jako jądro (GNU/Linux), ale jądro Hurd stale rozwijane
- assembler, kompilator GCC, glibc, program łączący, biblioteka GNU C, bash, GNU coreutils
- programy z projektów GNU <https://www.gnu.org/software/>
- GNU General Public License: <http://www.gnu.org/licenses/gpl.html>
Powszechna Licencja Publiczna GNU:
<http://www.gnu.org/licenses/gpl.html>
- Inne licencje: GNU Lesser GPL, GNU Library GPL, Modified BSD License, Perl Artistic License, Apache License, LaTeX Project Public License, Python Software Foundation License, PHP License, OpenSSL License, Sleepycat License, Common UNIX Printing System License Agreement, IBM Public License, . . .
- dystrybucje Linux: Slackware (1993), Debian (1993), Ubuntu (2004), RedHat, ...

comp.os.minix: wpis LBT z 25.08.1991

Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things).

I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them :-)

Linus (torvalds@kruuna.helsinki.fi)

PS. Yes - it's free of any minix code, and it has a multi-threaded fs. It is NOT portable (uses 386 task switching etc), and it probably never will support anything other than AT hard disks, as that's all I have :-(.

Rozwój GNU/Linux

Data	Wersja	Linie kodu	
1991/09	0.01	10 K (1K assemblera)	88 plików
1991/11	0.11		
1992/02	0.12		GNU GPL
1992/03	0.95		X Window
1994/04	1.0.0	176 K	
1995/03	1.2.0	311 K	arch. Alpha, SPARC i MIPS, szyna PCI
1996/06	2.0.0	470 K	procesory SMP, SMB
1999/01	2.2.0	1.8 M	
1999/12	2.2.13		
2001/01	2.4.0	3.4 M	ISA Plug-And-Play, USB, PCMCIA, arch. 64 bit, LVM
2003/12	2.6.0	5.9 M	NuMA, hiperwątkowość
2011/07	3.0	14.6 M	
2012/09	3.2	15.9 M	
2014/12	3.18	19.0 M	
2015/07	4.0	22.2 M	
2017/08	4.13	24.8 M	
2020/01	4.19	27.8 M	
2020/08	5.8	28 442 673	70K plików
2024/10	6.11.3		aktualna stabilna wersja

Rozwój linux (raport 2020)

- jeden z największych projektów programistycznych rozwijanych przez społeczność
- regularne wydania co 8-10 tygodni, każde z istotnymi zmianami i usprawnieniami
- tempo zmian jądra jest wysokie i rośnie, około 12 K łatek wchodzi obecnie do wydania (w 2020 r. 10.7 commit/h)
- 1600 programistów z ponad 200 korporacji
- ok 12% zmian od miłośników Linuksa
- ok 52% od programistów różnych firm, m.in. Intel, Red Hat, IBM, SUSE, Linaro, Google, Samsung, AMD, Renesas, Texas Instruments, and Oracle
- łącznie od 2005 ponad 15,600 indywidualnych programistów z 1400 różnych firm wniosło wkład w rozwój

- 1998 powstaje Open Source Initiative (Eric S. Raymond, prezes) i termin open source software w kontrze do free software
upublicznienie źródeł przeglądarki Netscape
Oracle, Informix, Sysbase, IBM, Dell, HP i in. ogłaszają wsparcie dla ruchu open source
- 2001 IBM – 1 mld USD i 1500 programistów wspiera rozwój Linuksa
Microsoft CEO S. Ballmer: „*Linux is a cancer [...] intellectual property destroyer*”
- 2000 SUSE Linux Enterprise Server (IBM S/390, x86), Novell 2004
- 2002 Red Hat Linux Advanced Server – pierwszy linuksowy system klasy enterprise wspierany przez firmy Dell, IBM, HP, Oracle
- 2007 projekt Samba otrzymuje (po procesie) od Microsoftu dokumentację SMB
- 2009 Microsoft dokłada ponad 20 tys. linii kodu do jądra Linuksa (Hyper-V)
- 2014 Satya Nadella (CEO firmy Microsoft): *Microsoft loves Linux*
- 2016 Windows Subsystem for Linux,
Microsoft przyłącza się do Linux Foundation jako platynowy sponsor

Współczesny rynek systemów operacyjnych

Komputery osobiste (PC, laptopy)

- Windows 73%
- OS X 15%
- Linux 4%
- Chrome OS 2%

Superkomputery

- Linux (TOP 500) 100%

Serwery (2021)

- Linux/Unix 77%
- Windows Server 22.7%

Tablety

- iOS 55%
- Android 44%

Mobilne

- Android 72%
- iOS 27%

Najważniejsze usprawnienia w architekturze komputerów

- koncepcja rodziny (IBM 360, PDP-8), kompatybilność architektury, skalowalność
 - mikroprogramowalna jednostka sterująca (IBM linia 360, 1964)
 - pamięć podręczna (model 85 IBM S/360, 1968); znaczące poprawienie wydajności
 - przetwarzanie potokowe
 - zwielokrotnione strumienie, multiplekser (IBM 370/168)
 - pobieranie docelowego rozkazu z wyprzedzeniem (IBM 360/91), bufor pętli (CDC 6600, CRAY-1)
 - przewidywanie rozgałęzienia (VAX 11/780, 10/1977)
 - wieloprocessorowość
 - architektura o zredukowanej liczbie rozkazów (RISC)
-

Ewolucja mikroprocesorów firmy Intel

parametr	8008	8080	8086	80386	80486
rok wprowadzenia	1972	1974	1978	1985	1989
liczba rozkazów	66	111	133	154	235
szerokość szyny adresowej	14	16	20	32	32
szerokość szyny danych	8	8	16	32	32
liczba rejestrów	8	8	16	8	8
adresowalność pamięci	16KB	64KB	1MB	4GB	4GB
szerokość pasma magistrali (MB/s)	-	0.75	5	32	32
czas dodawania rejestr-rejestr (μ s)	-	1.3	0.3	0.125	0.06

Mikroprocesory firmy Intel ¹⁰

parametr	286	386	486	Pentium	P6
początek projektowania	1978	1982	1986	1989	1990
rok wprowadzenia	1982	1985	1989	1993	1995
liczba tranzystorów	130K	275K	1.2M	3.1M	5.5M
szybkość (MIPS)	1	5	20	100	150

wieloprogramowość

supers
wielopro

¹⁰W.Stallings, *Organizacja i architektura systemu komputerowego*, WNT, Warszawa, 2000.

Prawo Moore'a (1965)

Gordon Moore – założyciel i wiceprezydent firmy Intel

Sformułowanie I:

Liczba tranzystorów, które można zmieścić na jednym calu kwadratowym płytki krzemowej podwaja się co 12 miesięcy.

Sformułowanie II:

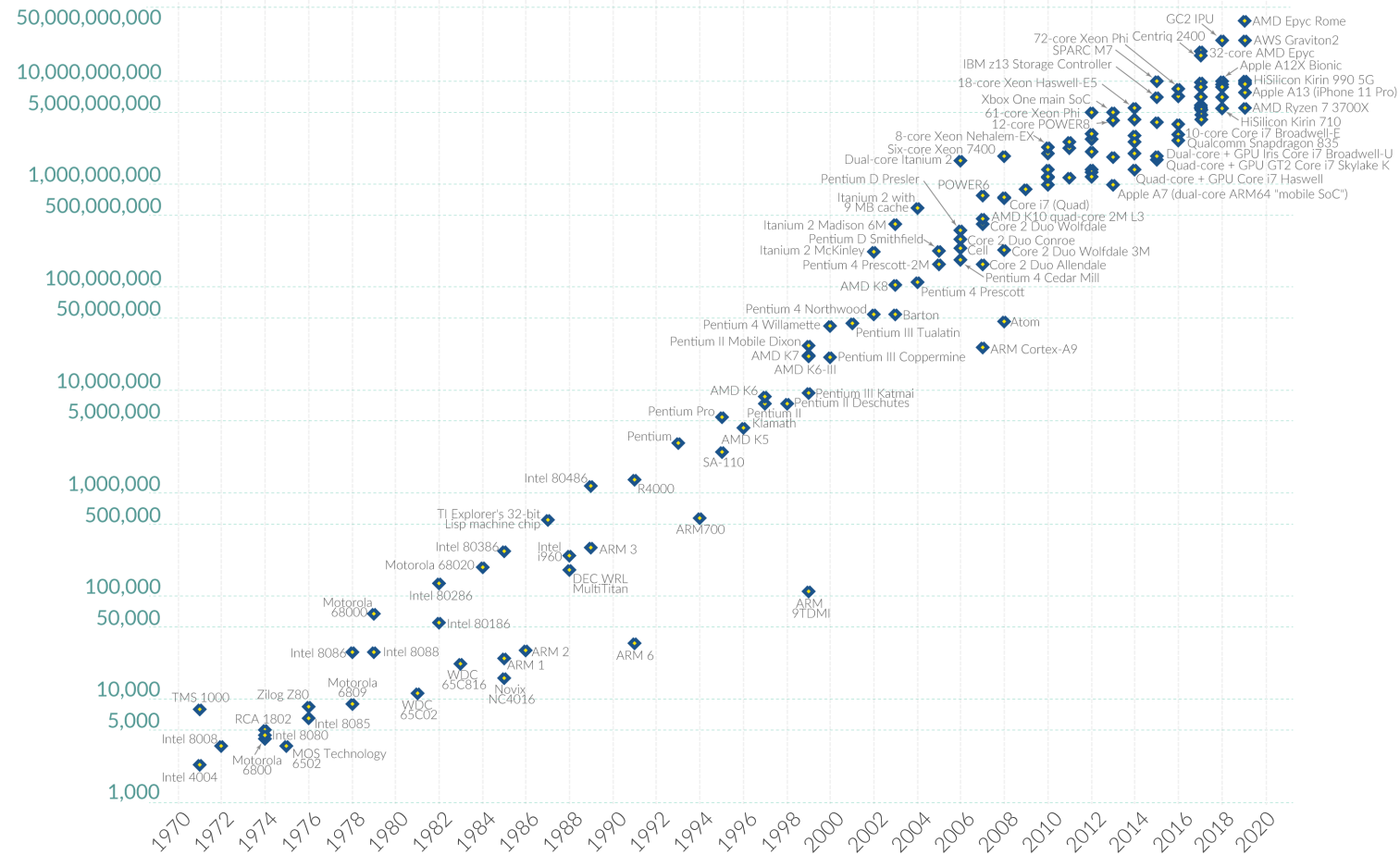
Liczba tranzystorów (na jednostce powierzchni płytki krzemowej), która prowadzi do najmniejszych kosztów na jeden tranzystor, podwaja się w przybliżeniu co 12 miesięcy.

Sformułowanie III:

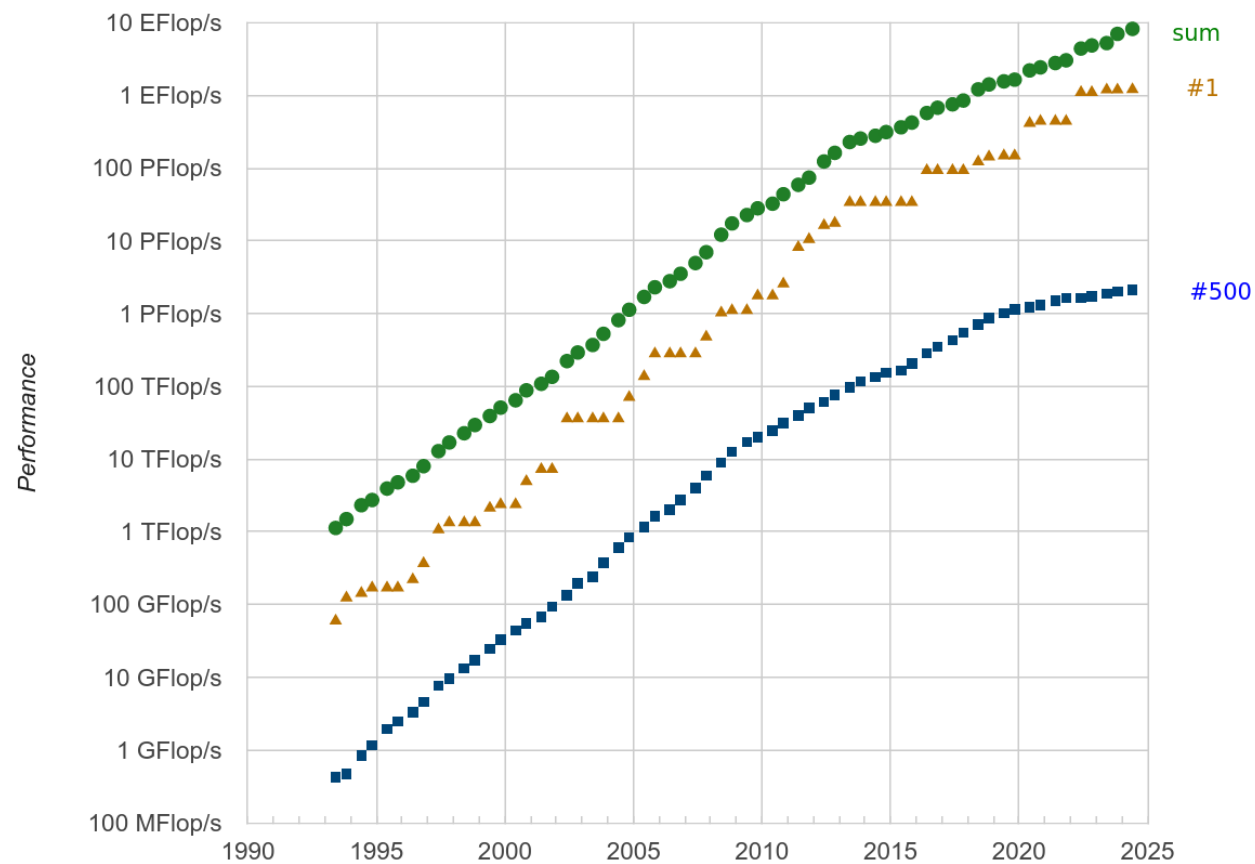
Wydajność systemów komputerów ulega podwojeniu co około 18 miesięcy.

Prawo Moore'a

Transistor count



TOP500: wzrost wydajności superkomputerów



Architektura współczesnego procesora

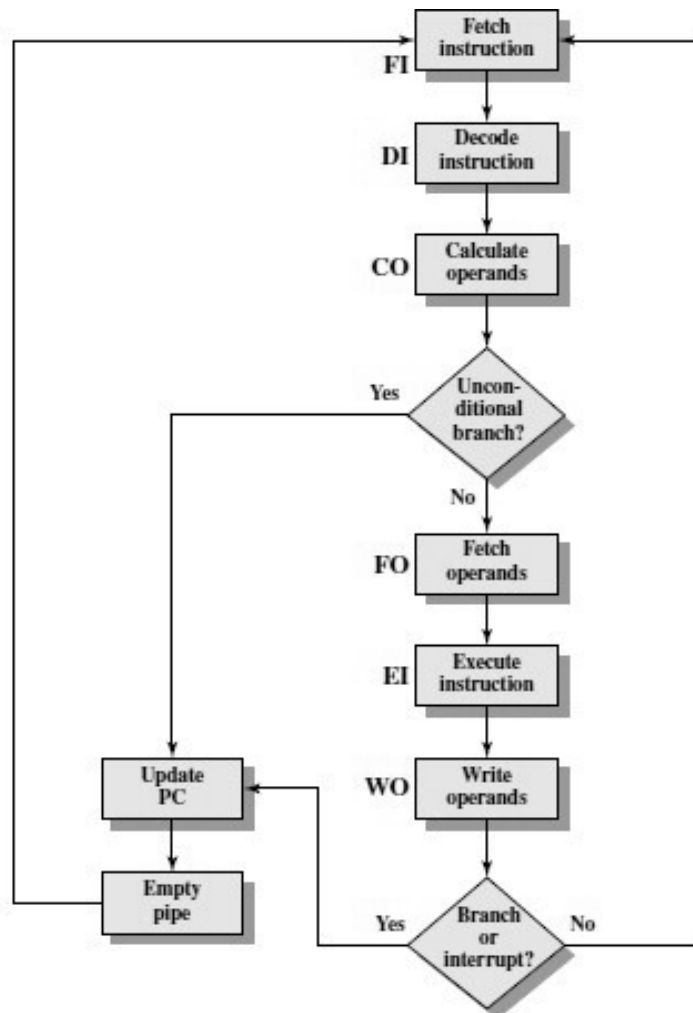
- przetwarzanie potokowe
- superskalarność
- przewidywanie rozgałęzienia (*branch prediction*), spekulatywne wykonywanie rozkazów
- analiza przepływu danych, tj. badanie zależności między rozkazami i wykonywanie ich nawet w kolejności innej niż w programie, aby zmniejszyć opóźnienia
- hiperwątkowość (*hyper-threading*)
- instrukcje SIMD (*Single Instruction Multiple Data*): MMX (*MultiMedia Extensions*), SSE (*Streaming SIMD Extensions*), AMD 3DNow (*3D NO Waiting*)
- wielordzeniowość
- wirtualizacja: Intel VT (vmx), AMD V (svm)
- GPGPU (*General-Purpose Graphical Processor Unit*)¹¹

¹¹CUDA ZONE http://www.nvidia.com/object/cuda_home.html

Processor

- rejestry
 - rejestry ogólnego zastosowania (x86: EAX, EBX, ECX, EDX)
 - licznik programu (PC)
 - wskaźnik bazowy (base pointer) (x86: EBP)
 - wskaźnik stosu (x86: ESP)
- słowo statusu programu (PSW *program status word*)
 - wyniki instrukcji warunkowych, flagi, bity kontrolne, stan systemu
 - tryb ochrony (tryb jądra, użytkownika)
- potok instrukcji

Przetwarzanie rozkazów



FI: pobierz następną instrukcję do bufora

DI: dekodowanie instrukcji, określ kod operacji i specyfikację argumentów

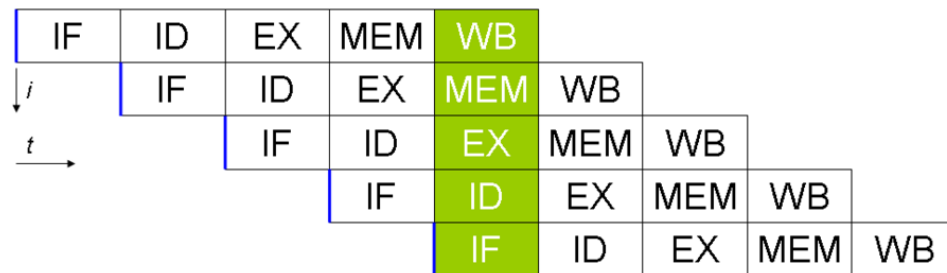
CO: oblicz efektywny adres danych źródłowych

FO: pobierz dane z pamięci jeżeli nie ma ich w rejestrach

EI: wykonaj instrukcję

WO: zapisz wynik w pamięci

Przetwarzanie potokowe



Klasyczny potok RISC:

IF=Instruction Fetch

ID=Instruction Decode

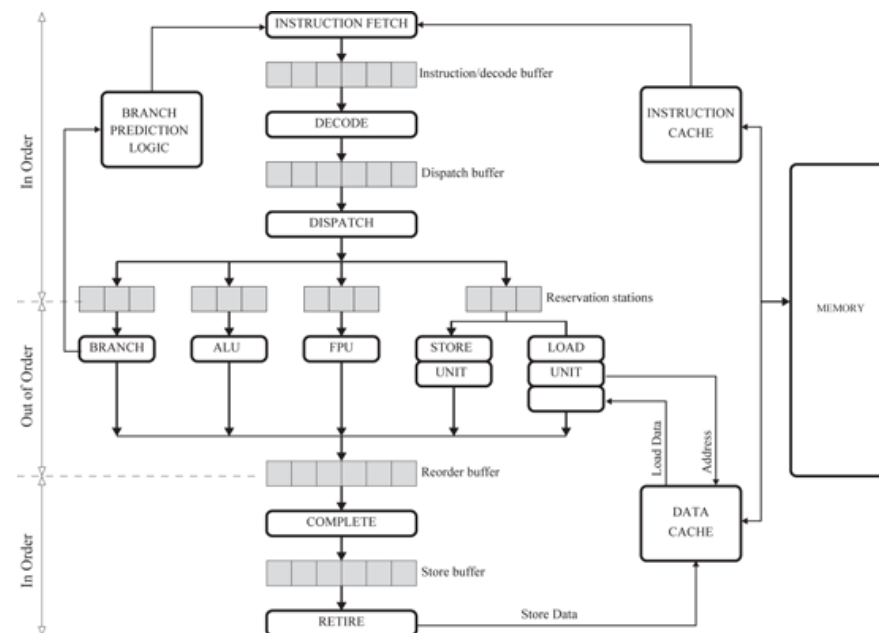
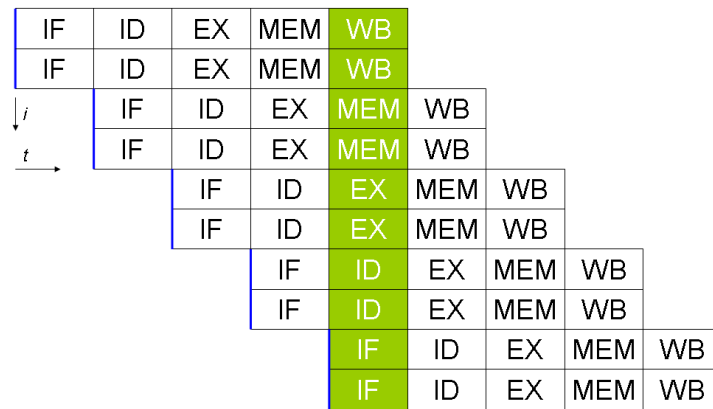
EX=Execute

MEM=Memory access

WB=Register write back

- podział rozkazu na wyspecjalizowane etapy
- dzięki podziałowi, różne instrukcje mogą być wykonywane równocześnie
- lepsza wydajność: zwiększenie liczby instrukcji wykonywanych w jednostce czasu
- problem z rozgałęzieniami (instrukcja skoku) i konflikty w dostępie do zasobów

Superskalarność



- zrównoleżenie wykonania rozkazów w obrębie jednego procesora poprzez zwielokrotnienie skalarnych jednostek wykonawczych
- w jednym cyklu procesora wykonywane jest wiele instrukcji
- wykonanie poza kolejnością (*out-of-order*) instrukcji z bufora w celu zmaksymalizowania użycia CPU

Ryzyka przetwarzania potokowego

Zatory w potoku poleceń pojawiają się, gdy instrukcje warunkowe nie pozwalają na sekwencyjne wykonanie instrukcji.

Możliwe ryzyka wykonania kodu poza kolejnością:

- **strukturalne** - gdy instrukcje w potoku nie mogą być wykonane równocześnie
- **danych** - gdy instrukcje korzystają z tych samych danych i wynik zależy od kolejności operacji
- **sterowania** - wykonanie instrukcji po rozwidleniu ale przed wykonaniem instrukcji warunkowej. Instrukcje z błędnej ścieżki muszą zostać odrzucone z potoku.

Ryzyka przetwarzania potokowego

Jak rodzić sobie z konfliktami:

- wykonywanie instrukcji obu ścieżek jednocześnie
- przewidywanie rozgałęzień (np. na podstawie statystyki poprzednich wykonań). Kod jest spekulatywnie wykonywany. W przypadku błędnej predykcji efekt tego wykonania jest odrzucany
- *brakch-free-code* technika programowania z minimalizowaniem rozgałęzień kodu
- bufor pętli - mała, bardzo szybka pamięć podręczna dla etapu pobierania (FI), jeśli wystąpi rozgałęzienie, sprzęt najpierw sprawdza, czy cel rozgałęzienia znajduje się w buforze pętli
- opóźnianie rozgałęzień - zmiana kolejności instrukcji tak aby instrukcje warunkowe można było wykonać jak najpóźniej

Własności procesorów CISC, RISC, superskalarnych (SS)

	CISC		RISC		SS	
	(a)	(b)	(c)	(d)	(e)	(f)
rok powstania	1978	1989	1988	1991	1990	1989
liczba rozkazów	303	235	51	94	184	62
rozmiar rozkazu [B]	2-57	1-11	4	32	4	4,8
tryby adresowania	22	22	3	1	2	11
liczba rejestrów	16	8	32	32	32	23-256
cache [KB]	64	8	16	128	32-64	0.5

(a) VAX 11/780, (b) Intel 80486, (c) Motorola 88000 (d) MIPS R4000, (e) IBM RS 6000, (f) Intel 80960

CISC (*Complex Instruction Set Computer*)

komputer o pełnej liście rozkazów

RISC (*Reduced Instruction Set Computer*)

komputer o zredukowanej liście rozkazów

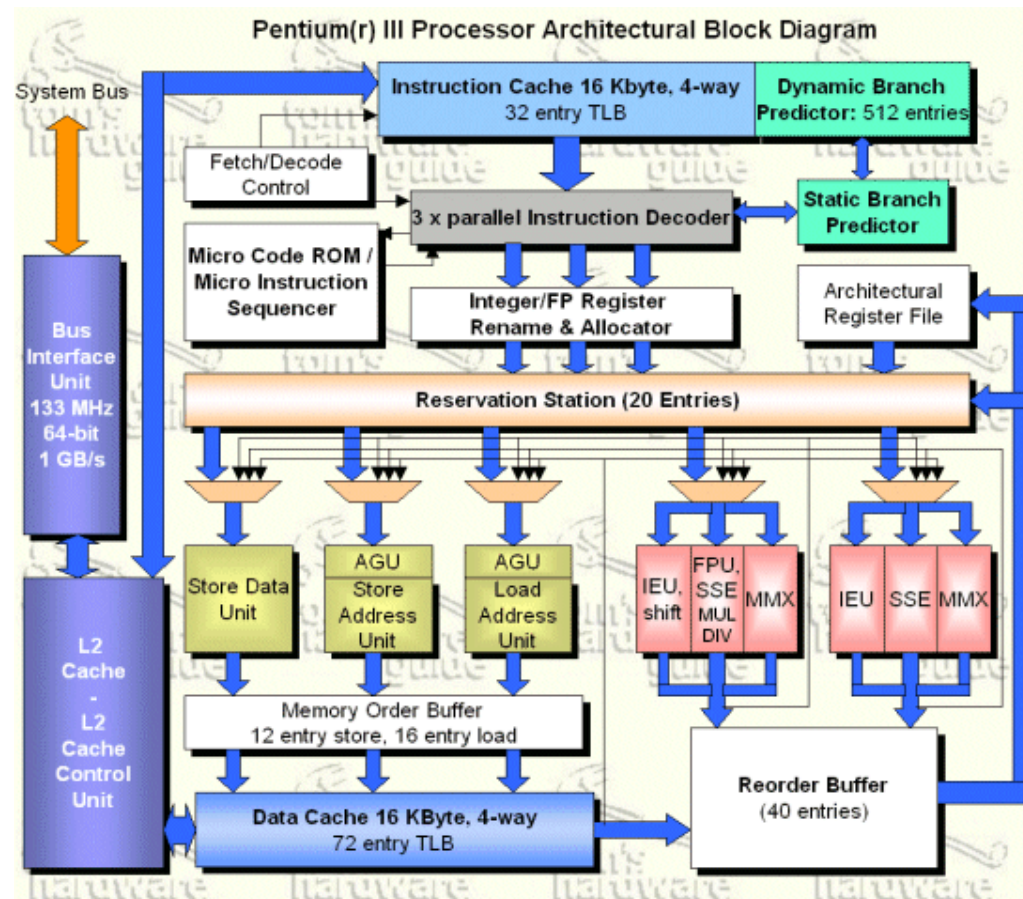
CISC (*Complex Instruction Set Computer*) komputer o pełnej liście rozkazów

- duża liczba rozkazów (100-200 instrukcji)
- złożone, wyspecjalizowane rozkazy
- wykonanie rozkazów wymaga dużej liczby cykli zegara
- duża liczba trybów adresowania (5-20)
- mikroprogramująca jednostka sterująca (ROM)

RISC (*Reduced Instruction Set Computer*) komputer o zredukowanej liście rozkazów

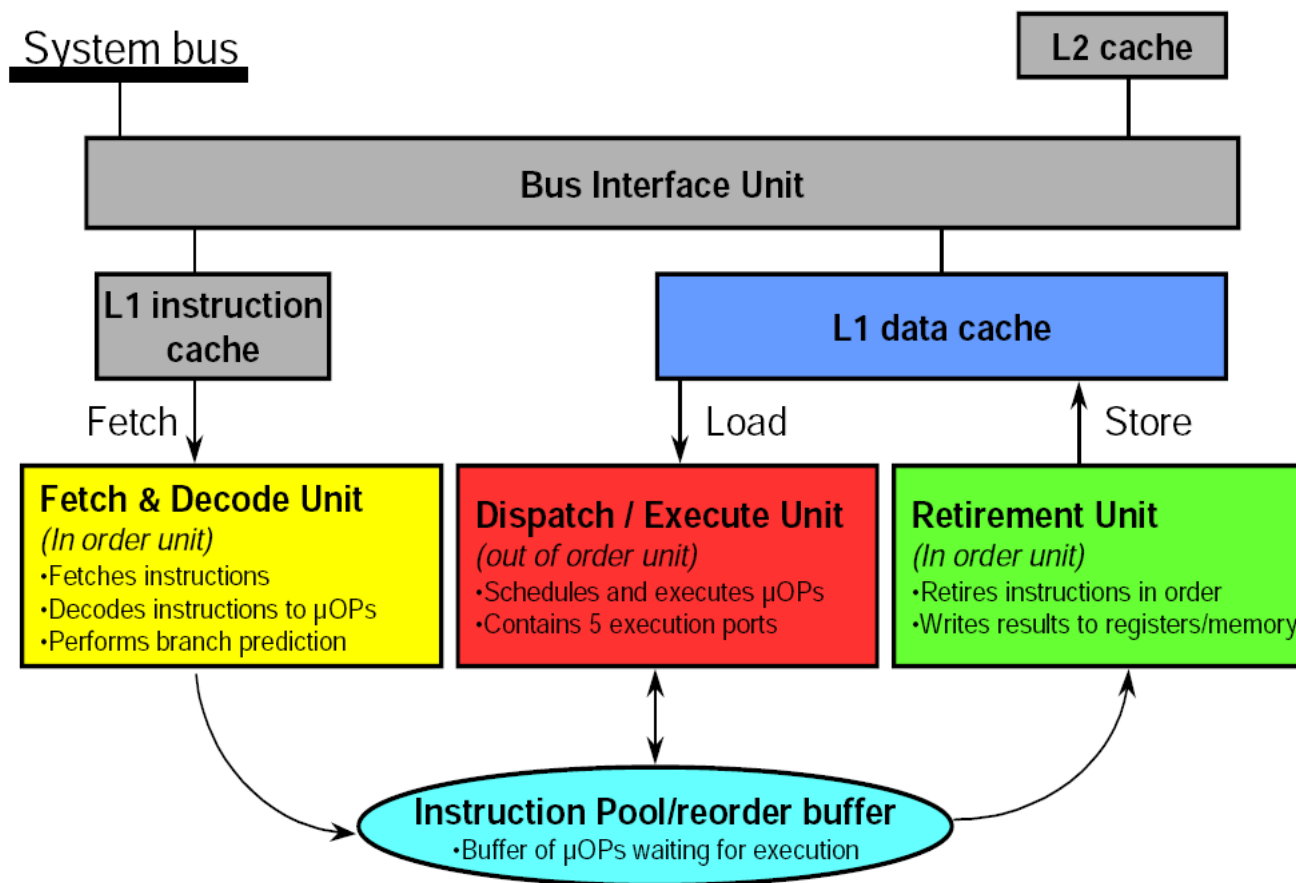
- zredukowany (mały) zestaw instrukcji (kilkadziesiąt)
- instrukcje wykonywane w pojedynczym cyklu zegara
- redukcja trybów adresowania
- rozkazy stałej długości (32 bity)
- ograniczenie odwołań do pamięci
- zwykle większa liczba rejestrów
- jednostka sterująca realizowana układowo
- prostsza architektura, ułatwia projektowanie procesorów

Obecnie architektura mieszana - złożone instrukcje rozkładane są na prostsze (μ OPS)



Zmodyfikowana architektura harwardzka architektura mieszana, łączy w cechy architektury harwardzkiej i architektury von Neumanna. Oddzielone obszary pamięci na dane i rozkazy.

Pentium II and Pentium III Processors Architecture



Pamięć: rodzaje, własności

- szybki rozwój pojemności pamięci i szybkości procesorów
- wolniejszy przyrost szybkości przesyłania danych pomiędzy procesorem i pamięcią

Interfejs pomiędzy pamięcią główną a procesorem jest najbardziej krytycznym elementem całego komputera, ponieważ jest on odpowiedzialny za przepływ rozkazów i danych pomiędzy tymi układami.

Jeśli dostęp do pamięci jest niewystarczający, to cykle procesora są marnowane (**głodzenie procesora**).

Wymagania dotyczące pamięci: niska cena, duża pojemność, szybkość działania (porównywalna z szybkością CPU)

Hierarchia pamięci:

- rejestry (SRAM)
- pamięć podręczna, cache (SRAM)
- pamięć główna (MCDRAM, DRAM, NVRAM, NVDIMM)
- pamięć dyskowa (HDD, SDD)
- pamięć taśmowa, dyski optyczne

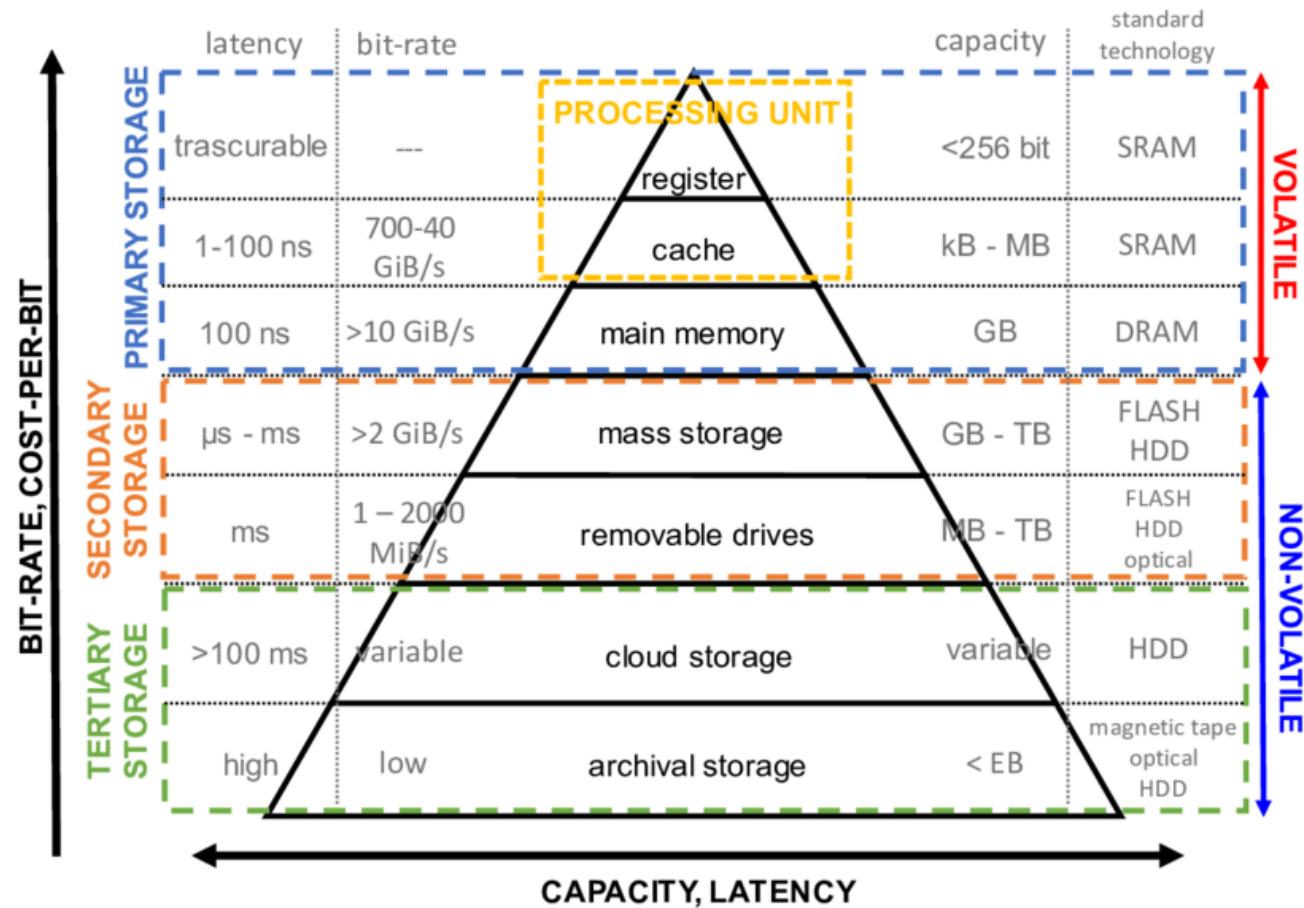
Koszt trafienia/chybień (w cyklach) dla procesora Pentium M:

rejestr	≤ 1
L1d	≈ 3
L2	≈ 14
pamięć główna	≈ 240

Zob.: [Interactive latency](#)

```
$ curl cheat.sh/latencies
```

Hierarchia pamięci¹²



¹²E. Gemo, *The design and analysis of novel integrated phase-change photonic memory and computing devices*, 2021

Hierarchia pamięci

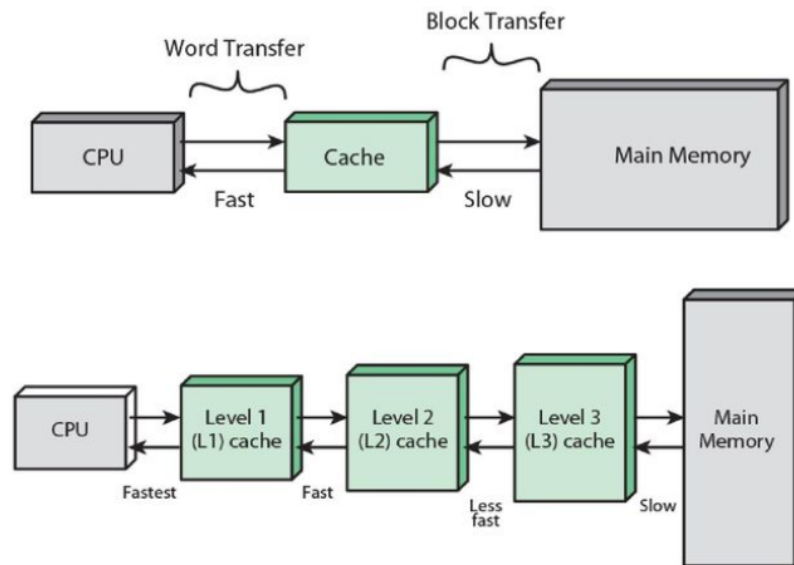
Im niżej w hierarchi od CPU

- malejąca cena/bit
- rosnąca pojemność
- rosnący czas dostępu
- malejąca częstość dostępu przez procesor

Rodzaj dostępu do pamięci:

- dostęp sekwencyjny
 - odczyt/zapis wyłącznie w określonej sekwencji liniowej (np. pamięć taśmowa)
 - czas dostępu zależy od położenia danych
- dostęp swobodny
 - pozycja w pamięci dostępna za pomocą unikatowego adresu (pamięć RAM)
 - czas dostępu stały, nie zależy od lokalizacji danych
- dostęp bezpośredni
 - pamięć podzielona na bloki (lub rekordy) o unikatowych adresach (np. pamięć dyskowa)
 - najpierw dostęp (swobodny) do najbliższego otoczenia (wybór bloku) a potem poszukiwanie loacji końcowej (sekwencyjne)
- dostęp skojarzeniowy
 - rodzaj dostępu swobodnego, który umożliwia porównywanie i specyficzne badanie zgodności wybranych bitów wewnątrz słowa (zachodzi to dla wszystkich słów jednocześnie)
 - słowo jest wyprowadzane na podstawie części swojej zawartości, a nie na podstawie adresu (zastosowanie: pamięć podręczna procesora)
 - stały czas dostępu

Pamięć podręczna



- przechowuje kopie fragmentu pamięci głównej
- procesor najpierw sprawdzana obecność danych w pamięci podręcznej
- jeśli brakuje (chybienie), to wczytywany jest blok pamięci głównej

Lokalność przestrzenna

- tendencja do częstych odwołań do danych położonych blisko siebie w pamięci
- takie dane mają szansę trafić do tego samego bloku w pamięci podręcznej

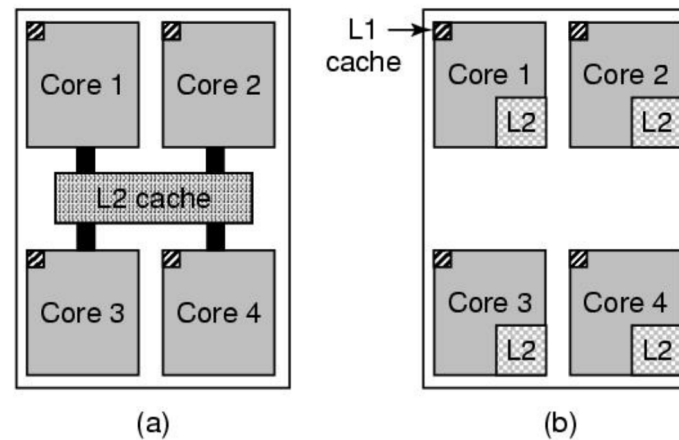
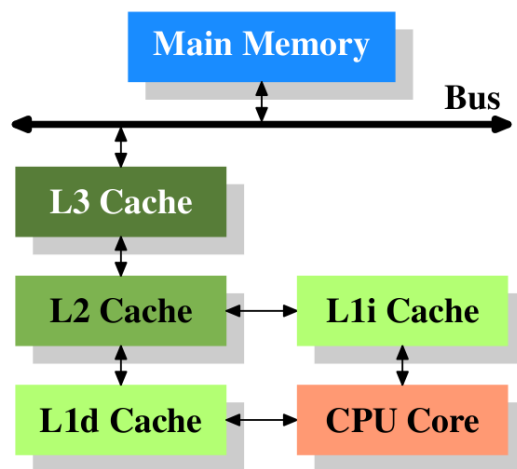
Lokalność czasowa

- tendencja do częstych odwołań do tych samych danych w krótkich odstępach czasu
- dane często wykorzystywane mają szansę pozostać w pamięci podręcznej pomiędzy rządzeniami

Procesor i pamięć

Architektura jednoprocessorowa

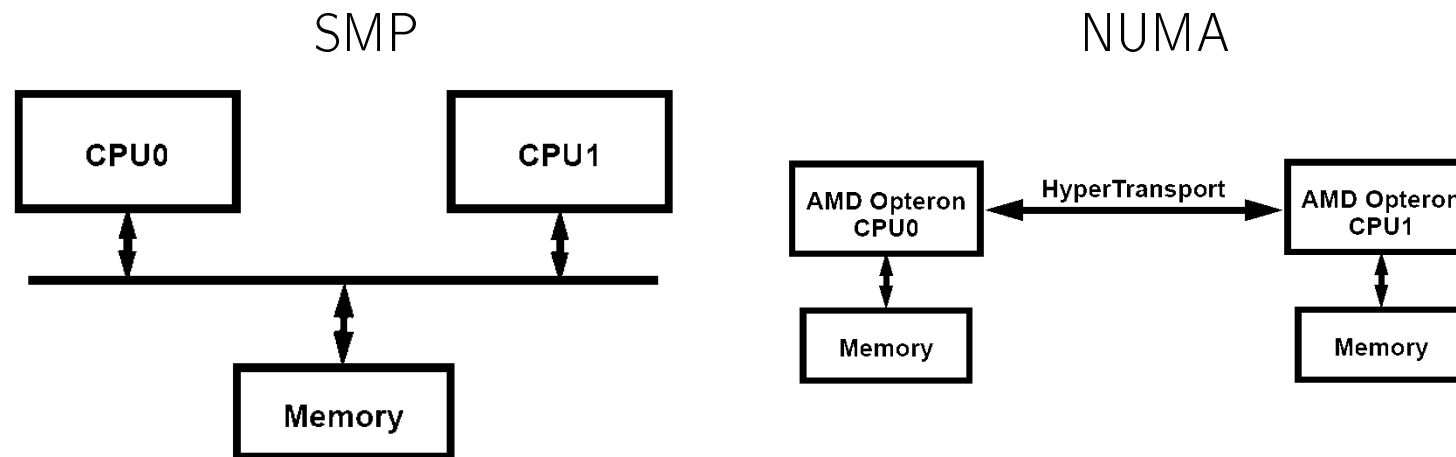
Architektura wieloprocessorowa



a) współdzielony L2 przez 4 procesory

b) niezależne L1 i L2 w układzie 4 procesorów

Procesor i pamięć



Wieloprocessorowość symetryczna

SMP *Symmetric Multiprocessing*

Niejednolity dostęp do pamięci

NUMA (*Non-Uniform Memory Access*)

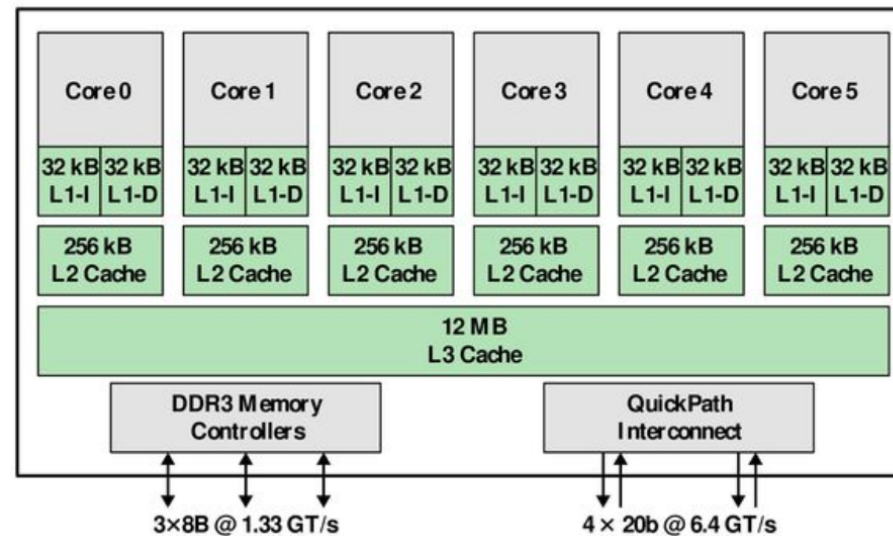
SMP

- procesory współdzielą zasoby pamięci oraz urządzenia wejścia/wyjścia przy pomocy magistrali systemowej
- każdy z procesorów może zostać przypisany do wykonywania konkretnego zadania, tak aby w systemie następowało równoważenie obciążenia (ang. load balancing)
- przepustowość szyny i pamięci ogranicza szybkość działania procesorów

NUMA

- minimalizacja opóźnień dostępu do pamięci poprzez przydzielenie pamięci lokalnej dla każdego procesora
- każdy procesor posiada własny kontroler pamięci lokalnej
- procesory łączone między sobą za pomocą szyny o dużej przepustowości i niskich opóźnieniach (HyperTransport), która nie ma bezpośredniego połączenia z układem pamięci
- wydłużony czas dostępu do pamięci zdalnej (połączonej z innym procesorem)
- wymaga właściwej organizacji danych w pamięci aby zapewnić efektywność

Architektury wieloprocessorowe



Rysunek 1: Intel Core-i7 990X

- wiele procesorów (rdzeni) na jednej kości
- każdy rdzeń posiada jednostkę wykonawczą, rejestry, lokalną pamięć podręczną
- rdzenie współdzielą pamięć L3
- rdzenie mogą być wielwątkowe

Rodzaje pamięci półprzewodnikowych:

RAM (*Random Access Memory*) pamięć o dostępie swobodnym – odczyt-zapis, wymazywanie/zapisywanie elektryczne na poziomie bajta

Static RAM (SRAM) - statyczna pamięć o dostępie swobodnym

- rejestry i pamięć podręczna
- ulotna, przechowuje dane tak długo, jak długo włączone jest zasilanie
- każdy bit przechowywany w układzie zbudowanym z 6 transystorów (mniejsza gęstość danych niż DRAM)

Dynamic RAM (DRAM) - pamięć dynamiczna

- pamięć główna komputera
- ulotna, wymaga okresowego odświeżania zawartości

Pamięci dynamiczne

Synchronous DRAM (SDRAM) - pamięć dynamiczna pracująca synchronicznie z magistralą systemową

	napięcie	pojemność	przepustowość	rok
<i>single data rate</i>				
SDR SDRAM	3.3 V	512 MB	533-1066 MB/s	1996
<i>double data rate</i>				
DDR SDRAM:	2.5 V	1 GB	1.6-3.2 GB/s	2000
DDR2 SDRAM	1.8 V	4 GB	3.2-6.4 GB/s	2003
DDR3 SDRAM	1.5 V	8 GB	6.4-19.2 GB/s	2007
DDR4 SDRAM	1.2 V	16 GB	12.8-25.6 GB/s	2012
DDR5 SDRAM	1.1 V	64 GB	25.6-64 GB/s	2020

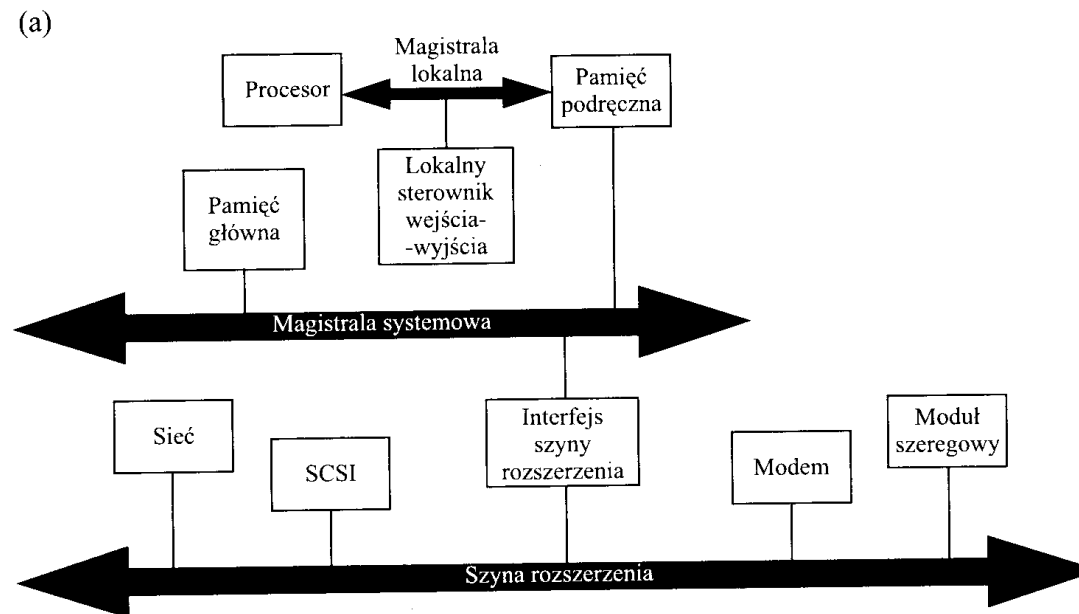
$$\text{moc} = \text{pojemność} \times \text{napięcie}^2 \times \text{częstotliwość}$$

Rodzaje pamięci półprzewodnikowych nieulotnych

- **ROM** (*Read-Only Memory*) pamięć stała – tylko odczyt, zapisywanie w trakcie produkcji
- **PROM** (*Programmable ROM*) programowalna pamięć jednokrotnego zapisu
- **EPROM** (*Erasable and Programmable ROM*) wymazywalna i programowalna pamięć stała – głównie odczyt, wymazywanie światłem UV, zapisywanie elektryczne
- **pamięć błyskawiczna** (*flash memory*) – wymazywanie elektryczne na poziomie bloku (256 B-16 KB), zapisywanie elektryczne; szybki odczyt, wolniejszy zapis (SSD, pamięci USB, karty pamięci MC, MMC, SM)
- **NVRAM** (*Non-Volatile RAM*) – pamięć typu SRAM/DRAM z baterią podtrzymującą zawartość po odłączeniu zasilania (zastosowanie BIOS)¹³
- **NVDIMM** (*Non-Volatile Dual In-line Memory Module*) – pamięć typu DRAM powiązana z układami pamięci błyskawicznej, co pozwala zachować zawartość pamięci ulotnej po awarii zasilania

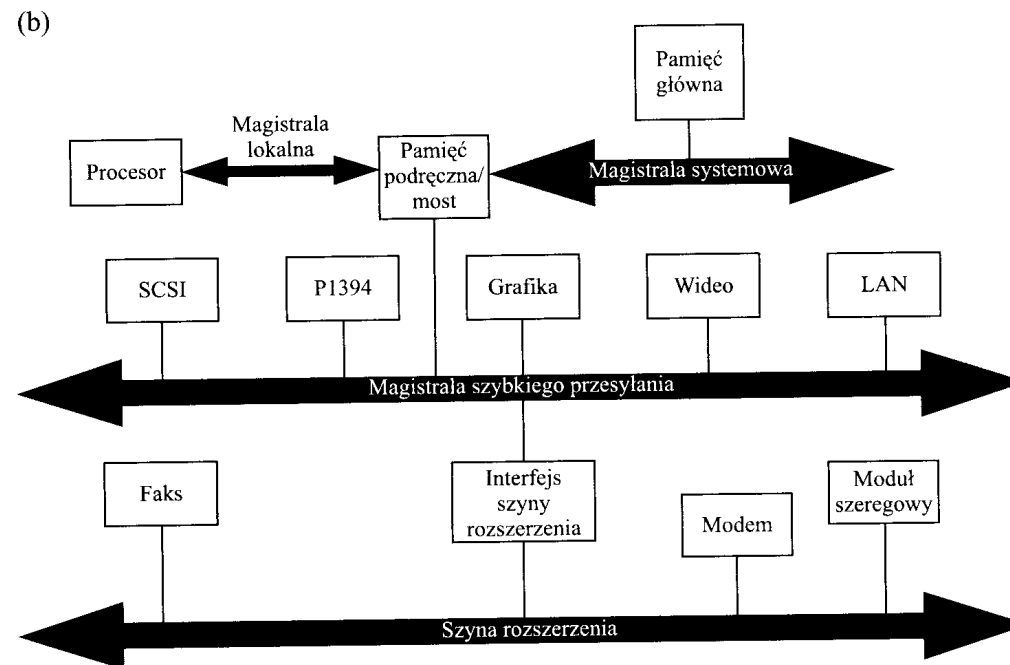
¹³Także EEPROM (*Electrically Erasable and Programmable ROM*) wymazywalna i programowalna pamięć stała (na poziomie bajtu) – głównie odczyt, wymazywanie/zapisywanie elektryczne; szybki odczyt, wolny zapis.

Magistrale (szyny) systemowe



- komunikacja między szyną systemową i szyną rozszerzenia może następować bez angażowania CPU

Magistrale (szyny) systemowe



- hierarchiczna magistrala o wysokiej wydajności
- urządzenia o większych wymaganiach są bliżej CPU

Magistrala (E)ISA

ISA (*Industry Standard Architecture*, IBM 1980) architektura standardu przemysłowego

- 8 MHz, szerokość 8 lub 16 bitów, przepustowość 8 MB/s
- często wymagała ręcznej konfiguracji IRQ, adresów I/O oraz kanału DMA

EISA (*Extended ISA*, Gang of Nine 1989) rozszerzona architektura standardu przemysłowego

- 8 MHz, szerokość 32-bitów, przepustowość 33 MB/s
- do 4 GB RAM
- *Plug and Play*, przez problemy z konfliktami IRQ nazywany *plug and pray*

Magistrala PCI

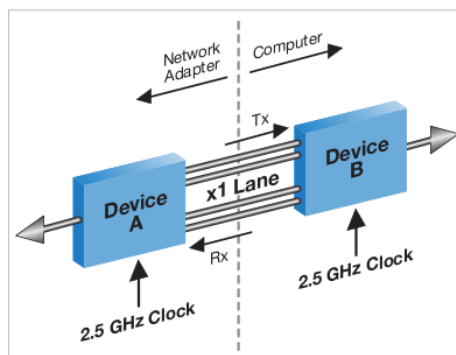
PCI (*Peripheral Component Interconnect*) interfejs komponentów peryferyjnych:

- 32/64-bitowa szyna rozszerzeń dla komputerów zgodnych z IBM PC oraz Macintosh
- opracowana przez firmę Intel w 1992 r. (mikroprocesor Pentium)
- obsługuje standard podłącz i używaj (PnP, *Plug and Play*)
- szyna równoległa, synchroniczna z zegarem
- każde urządzenie na osobnej szynie z osobnym adresem
- autokonfiguracja, system odpytuje PCI przy starcie o dostępne urządzenia

Magistrala PCI Express

- szyna szeregowowa o pełnym duplexie
- 1, 2, 4, 8 lub 16 linii
- połączenie Point-to-Point (podobnie jak HyperTransport) pozwalające na przesyłanie danych z dużą prędkością i instalację kart rozszerzeń na płycie głównej
- każde urządzenie jest połączone bezpośrednio z kontrolerem
- zastępuje PCI i AGP (Advanced Graphics Port, PCI dla kart graficznych)

Figure 3. PCI Express* x1 lane. A lane consists of two differentially driven signal pairs between two PCI Express devices (Device A and Device B).

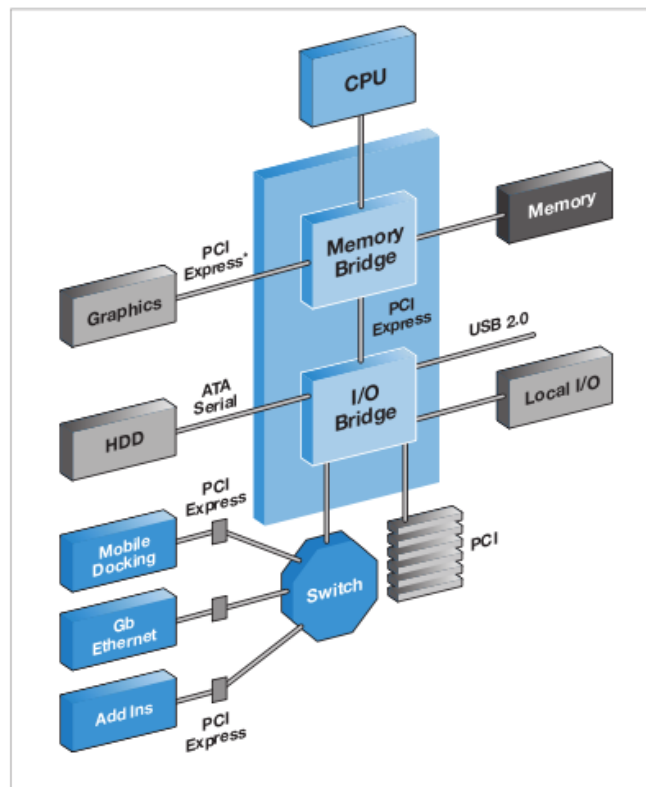


PCI i PCIe

wersja	szyna	częstotliwość	przepustowość	rok
PCI 2.0 (PCI32)	32 b	33.33 MHz	133.3 MB/s	1993
PCI 2.1 (PCI64)	64 b	33.33 MHz	266.6 MB/s	1994
PCI 2.2	32 b	66.66 MHz	266.6 MB/s	1999
PCI 2.3	64 b	66.66 MHz	533.3 MB/s	2002
PCI-X 1.0	64 b	133.3 MHz	1066.4 MB/s	1999
PCI-X 2.0	64 b	533.3 MHz	4266.4 MB/s	2002
PCI-X 3.0	64 b	1066.6 MHz	7.95 GB/s	2004
			przepustowość	
	kodowanie		jednokierunkowa	
PCIe 1.0 (×1)	8b/10b	2.50 GHz	250.0 MB/s	2004
PCIe 1.0 (×16)	8b/10b	2.50 GHz	4.0 GB/s	
PCIe 2.0 (×16)	8b/10b	5.00 GHz	8.0 GB/s	2007
PCIe 3.0 (×16)	128b/130b	8.00 GHz	16 GB/s	2011
PCIe 4.0 (×16)	128b/130b	16.00 GHz	32 GB/s	2017
PCIe 5.0 (×16)	128b/130b	32.00 GHz	64 GB/s	2019
PCIe 6.0 (×16)	1b/1b (tryb Flit)	64.00 GHz	128 GB/s	2021

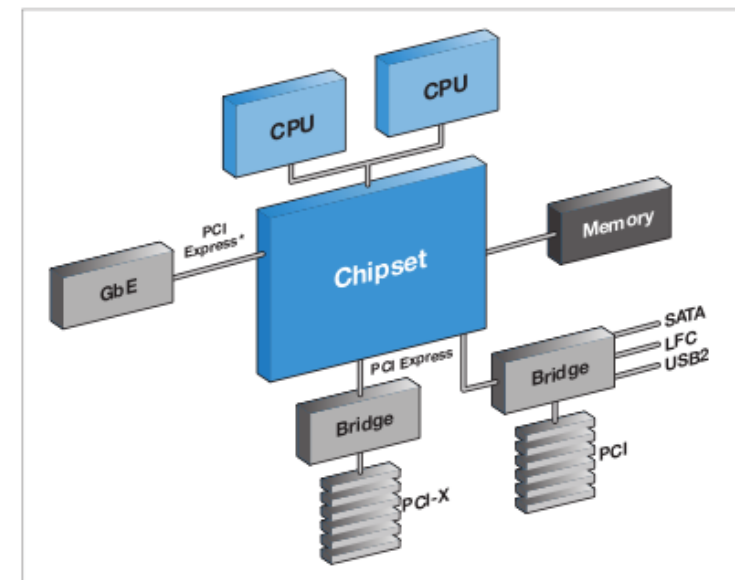
Magistrala PCI Express

Figure 4. Gigabit Ethernet to the desktop implemented through the high-speed serial I/O bus of PCI Express*.



płyta główna PC

Figure 5. Example of PCI Express* in server/workstation systems. The PCIe* switch is integrated into the chipset.



płyta główna serwera

Uniwersalna szyna szeregową (*Universal Serial Bus*, USB)

- standard szyny zewnętrznej do podłączania do komputera do 127 urządzeń peryferyjnych (jeden IRQ)
- standard opracowany w 1995 r. wspólnie przez wiodących producentów sprzętu komputerowego i telekomunikacyjnego (Compaq, DEC, IBM, Intel, Microsoft, NEC, Northern Telecom, Philips)
- wyprowadzenie funkcji PnP poza komputer, automatyczne wykrywanie sprzętu
- *hot plugging* - podłączenie nie wymaga wyłączenia zasilania
- łatwość rozmnażania portów i wydłużania połączenia poprzez zastosowanie maksymalnie pięciu koncentratorów (*USB hubs*),
- wymaga obecności dokładnie jednego kontrolera po stronie hosta (niemożliwe podłączenie dwóch komputerów)
- załączce uniwersalne: łączy drukarki, skanery, kamery wideo, dyski, stacje dyskietek, klawiatury, myszy, joysticki, telefony, modemy, itp.

Uniwersalna szyna szeregową

wersja	przepustowość		uwagi	rok
USB 1.1 low speed	1.5 Mb/s	0.1875 MB/s	półduplex, kabel 3m.	1998
full speed	12 Mb/s	1.5 MB/s		
USB 2.0 high speed	480 Mb/s	60 MB/s	kabel 5m.	2000
USB 3.0 super speed	5 Gb/s	625 MB/s	pełen duplex, kabel 3m.	2008
USB 3.1 super+ speed	10 Gb/s	1.25 GB/s		2013
USB 3.2	20 Gb/s	2.5 GB/s		2017
USB4	40 Gb/s	5.0 GB/s		2019
USB4 2.0	80 Gb/s	10.0 GB/s		2022

Interfejsy dysków: IDE/ATA

IDE (*Integrated Drive Electronics*) właściwie **ATA** (*Advanced Technology Attachment*), od 2003 – **PATA** (*Parallel ATA*)

- szyna do łączenia dysków HDD i napędów CDROM (40 pinów)
- IDE ATA-1, 1986, EIDE (*Extended IDE*), 1994
- UltraDMA (DMA-33, Ultra33, ATA-33) – od 1997 wydajność 33 MB/s (16 MB/s), najnowsze napędy CD-ROM and CD-RW
- ATA-66 (Ultra66, DMA-66) – od 1999
- ATA-100, ATA-133 – 80 pinów, przepustowość do 133 MB/s (prakt. ≈ 20 MB/s)
- ATAPI – *ATA Packet Interface* dla napędów CD, DVD, ...
- MTBF $\approx 1.2 \times 10^6$ h ¹⁴ (prawd. awarii dysku w czasie roku $\frac{24h \cdot 356}{MTBF} \approx 7\%$)
- zastąpiony przez SATA

Ograniczenia: każdy sterownik IDE potrzebuje IRQ i może obsłużyć 2 urządzenia wewnętrzne (*master* i *slave*); maks. dł. kabla – 475 mm

<https://itigic.com/pl/ide-ata-pata-and-atapi-interface-how-does-it-work/>

¹⁴MTBF (*Mean Time Between Failures*) – czas po jakim z pewnej puli urządzeń zostaje połowa sprawnych. Z 1000 dysków o MTBF 1.2×10^6 h będzie się psuł jeden co (średnio) $\frac{MTBF}{24h \cdot 1000} = 50$ dni.

Szeregowy ATA (*Serial ATA, SATA*) - najnowsza wersja IDE

- SATA I, szybkość (teor.): 150 MB/s , 2003 r.
- SATA II, 300 MB/s, (prakt.): ≈ 40 MB/s, 2004 r.
 - kolejkovanie zadań w celu minimalizacji liczby skosków głowicy
 - powielacze portów - możliwość podłączenia SATA do wielu urządzeń
 - wyznacznik portu - możliwość podłączenia dwóch portów do tego samego SATA (nadmiarowa ścieżka)
- SATA III, 600 MB/s, (prakt.) 170 MB/s, 2009 r.
- połączenia punkt-punkt
- możliwość podłączania wewnętrznych i zewnętrznych urządzeń
- CRC (*Cyclic Redundancy Check*) dla pakietów danych, rozkazów i statusu
- łatwość przyłączenia (wsparcie dla *hot swapping* - wymiana urządzenia bez rerestartu systemu)
- MTBF $\approx 2.5 \times 10^6$ h

Dyski SSD (*Solid State Drive*)

- Problem skończonej liczby cykli P/E
Żywotność współczesnych dysków jest rzędu kilkuset lat!
- SATA III 6Gbps
 - szybkość 460-560 MB/s¹⁵
- PCIe 4.0
 - szybkość 2500-7000 MB/s
- NVMe (*Non-Volatile Memory Express*)
specyfikacja interfejsu dla dysków SSD połączonych magistralą PCIe
 - szybkość 3000-3500 MB/s

¹⁵Wyznaczanie szybkości operacji dyskowych: `hdparm -tT /dev/sda`.

SCSI (*Small Computer Systems Interface*)

- równoległa magistrala dla stacji roboczych i serwerów (odpowiednik ATA w PC ale mniej obciążający CPU)
- jeden IRQ, obsługa do 16 urządzeń (w tym sterownik), łączy wielopunktowe (*multidrop*) dyski, napędy CD-ROM, napędy taśmowe (*streamer*), skanery
- maksymalna długość do 25 m, wsparcie do 16 urządzeń (Ultra SCSI)
- MTBF $\approx 1.6 \times 10^6$ h

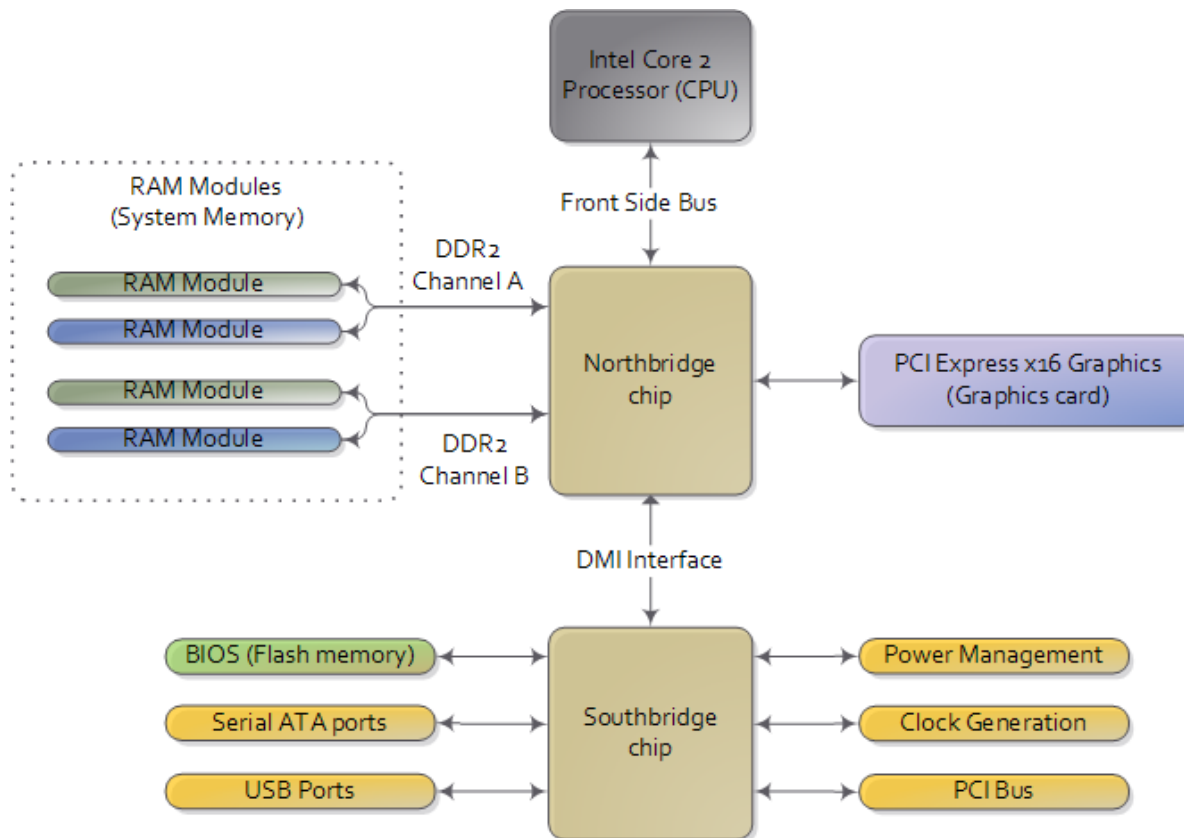
	szybkość	szer. (bity)	przepustowość	prakt.	rok
SCSI	5 MHz	8	5 MB/s	3 MB/s	1986
Fast SCSI	10 MHz	8	10 MB/s	8 MB/s	1994
Fast & Wide SCSI	10 MHz	16	20 MB/s	16 MB/s	
Ultra	20 MHz	8	20 MB/s		1996
Ultra 2	40 MHz	8	40 MB/s		1997
Ultra 160	80 MHz	16	160 MB/s	30-60 MB/s	1999
Ultra 320	160 MHz	16	320 MB/s	60-120 MB/s	2002
Ultra 640	160 MHz	16	640 MB/s		2003

SAS (*Serial Attached SCSI*)

- szeregową szyną, zastąpiła SCSI
- jeden IRQ, połączenie punkt-punkt (*initiator-target*), tryb pełnego duplexu
- zastosowanie ekspanderów (*expander*) daje możliwość podłączenia do 65535 urządzeń
- MTBF $\approx 1.6 \times 10^6$ h

	przepustowość	rok
SAS 1	3.0 Gb/s (300 MB/s)	2004
SAS 2	6.0 Gb/s (600 MB/s)	2009
SAS 3	12.0 Gb/s (1200 MB/s)	2013
SAS 4	22.5 Gb/s (2400 MB/s)	2017

Architektura płyty głównej



Chipset

- **Front Side Bus** - magistrala wyjściowa procesora
Komunikacja procesora z resztą świata:
 - adresy pamięci RAM
 - adresy I/O mapowane na adresy pamięci (Memory Mapped I/O)
 - przerwania i sygnały
- **mostek północny** - łączy CPU z urządzeniami dużej szybkości:
RAM, GPU, PCIe, Gigabit Ethernet
- **mostek południowy** - współpraca z pozostałymi urządzeniami
ICH (I/O Controller Hub) Intel, FCH (Fusion CH) AMD
- elementy chipsetu: sterownik CPU, sterownik DRAM, sterownik cache, sterownik przerwań, sterownik DMA, sterownik magistrali,
- mostek północny obecnie często zintegrowany z południowym lub nawet z CPU

Jak system wykonuje zadania?

- Przetwarzanie wsadowe, monitor prosty, praca pośrednia
- Buforowanie, Spooling
- Wieloprogramowość
- Wielozadaniowość, wielodostępowość, systemy z podziałem czasu
- Wieloprocessorowość, systemy HPC
- Systemy rozproszone
- Systemy czasu rzeczywistego

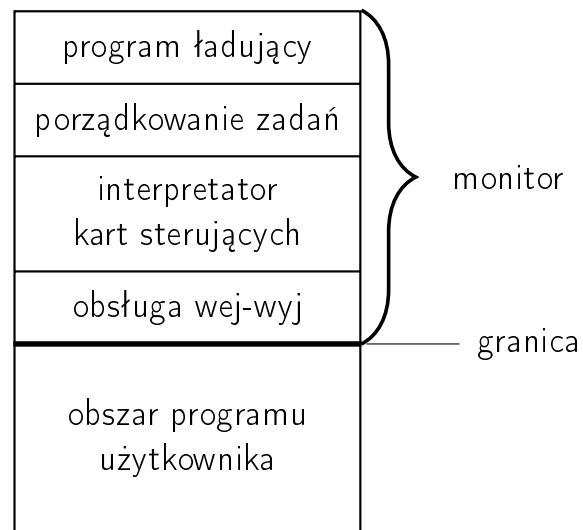
Monitor prosty

W czasie instalowania taśm jednostka centralna była bezczynna. Komputery były drogie i ich czas był cenny.

Rozwiązanie:

- zatrudnienie profesjonalnych operatorów
- **przetwarzanie wsadowe** (*batch processing*) - zadania gromadzone (przez operatora) w postaci wsadów są kolejno wykonywane, bez interakcji użytkownika
- automatyczne porządkowanie zadań (*automatic job sequencing*)
- **monitor rezydujący** – stale obecny w pamięci, automatyczne przekazywanie sterowania od zadania do zadania; karty sterowania zadaniami (JCL, *Job Control Language*)

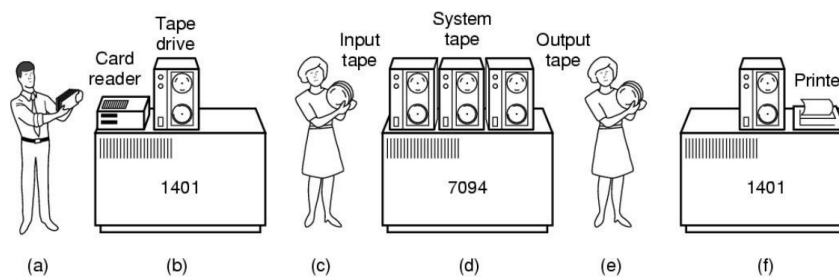
Monitor rezydujący w pamięci



- ochrona pamięci - program nie może ingerować w obszar pamięci monitora
- uprzywilejowane instrukcje - pewne instrukcje mogą być wykonywane wyłącznie przez monitor (tryb jądra)

Praca pośrednia (off-line)

- Wolne operacje wej-wyj wykonywane przez mniejsze, satelitarne minikomputery.
- Jednostka główna czyta dane z taśmy magnetycznej i umieszcza wyniki na taśmie magnetycznej.
- Obsługa czytników kart i drukarek wierszowych w trybie pośrednim (*off-line*).
- Niezależność od urządzeń wej-wyj – programy pisane z myślą o korzystaniu z logicznych, a nie fizycznych urządzeń peryferyjnych.



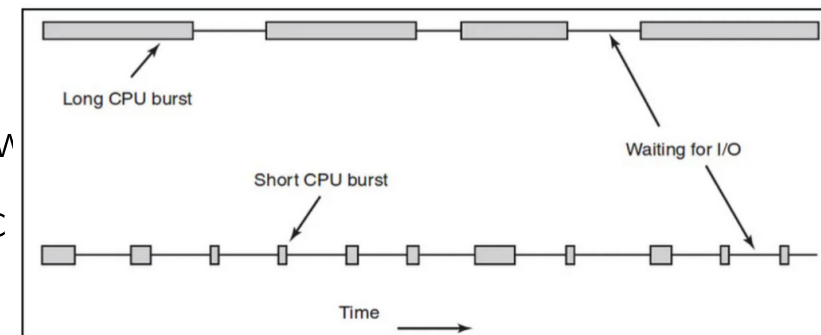
Buforowanie

Buforowanie jest metodą usprawniającą wykonywanie obliczeń i operacji wej-wyj dla danego zadania

- dane są tymczasowo przechowywane w pamięci (buforze) przed ich dalszym przetwarzaniem lub przesyłaniem
- buforowanie zmniejsza liczbę operacji I/O, przyspiesza dostęp do danych
- procesor może pracować nad innymi zadaniami, podczas gdy dane są ładowane do bufora. To pozwala na lepsze wykorzystanie zasobów procesora i zmniejsza czas oczekiwania na dane.

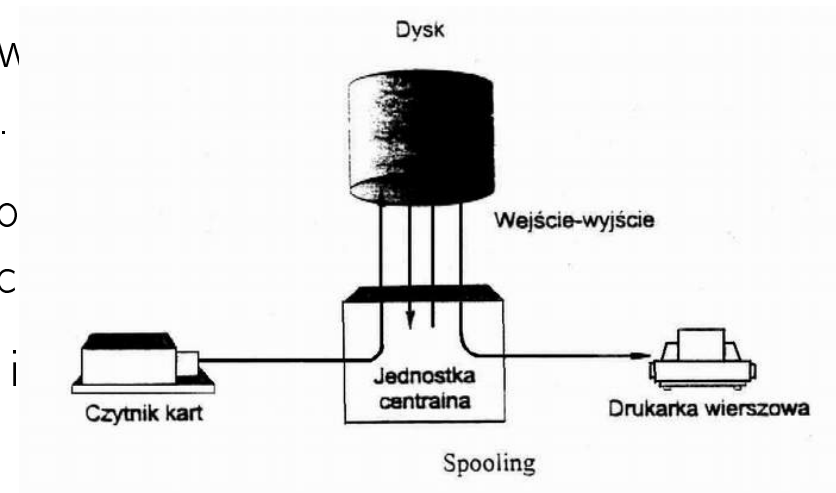
Zadania ograniczone CPU i ograniczone I/O

- Zadania ograniczone operacjami wej-wyj (*I/O bound*)
 - prędkość przetwarzania zależna od szybkości urządzeń wej-wyj.
 - buforowanie może zwiększyć przepustowość poprzez grupowanie wielu operacji I/O
- Zadania ograniczone procesorem (*CPU bound*)
 - prędkość przetwarzania ograniczona przez wejściowy zawsze pełny, bufor wyjściowy zaw
 - buforowanie zwiększa wydajność gromadząc zmniejszając czas dostępu



Spooling

- **spooling** (*Simultaneous Peripheral Operation On-Line*) jednoczesna bezpośrednia praca urządzeń
- dane są przechowywane w plikach na dysku w przetwarzania przez urządzenia peryferyjne (np.
- spooling umożliwia równoczesne wykonywanie o zadania i obliczeń dla innych zadań znajdującyc
- spooling wytwarza pulę zadań do wykonania i zadań i wieloprogramowość



Wieloprogramowość

- praca pośrednia, buforowanie, spooling mają swoje ograniczenia. Jeden użytkownik nie jest w stanie angażować stale urządzeń wej-wyj i jednostki centralnej!
- **wieloprogramowość** - w pamięci rezyduje więcej niż jeden program użytkownika, w momencie zablokowania (operacja I/O) CPU jest przełączany do innego zadania
- realizacja wieloprogramowości wymaga skomplikowanego systemu operacyjnego: ochrona zadań między sobą i planowanie przydziału procesora, kolejki, partycje o stałych lub zmiennych wielkościach

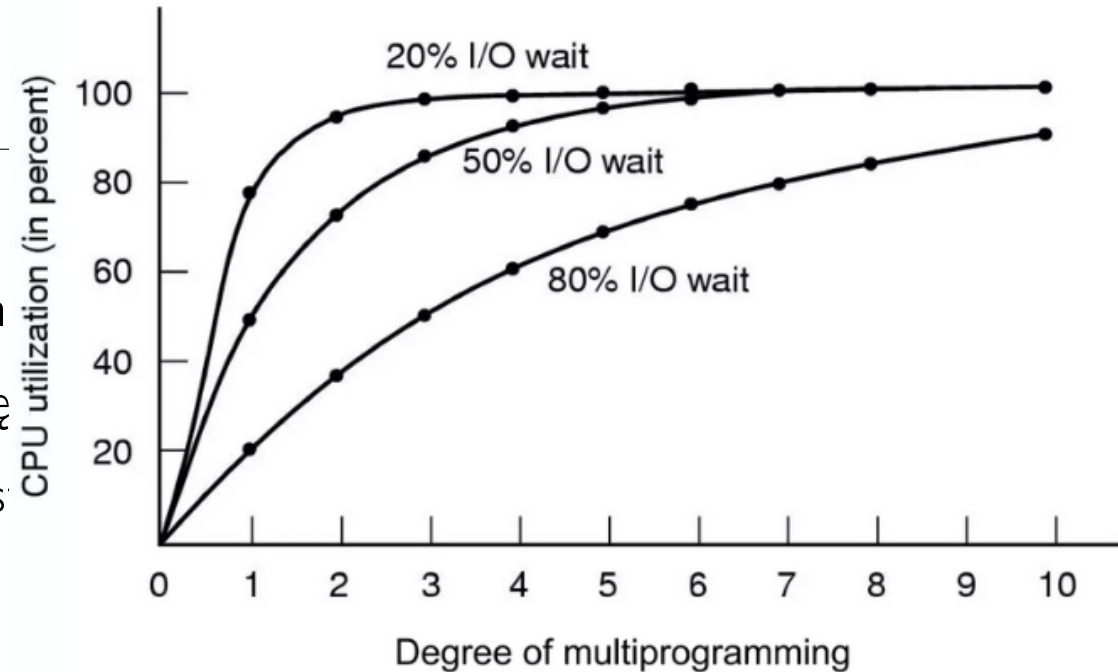


Jak system wykonuje zadania?

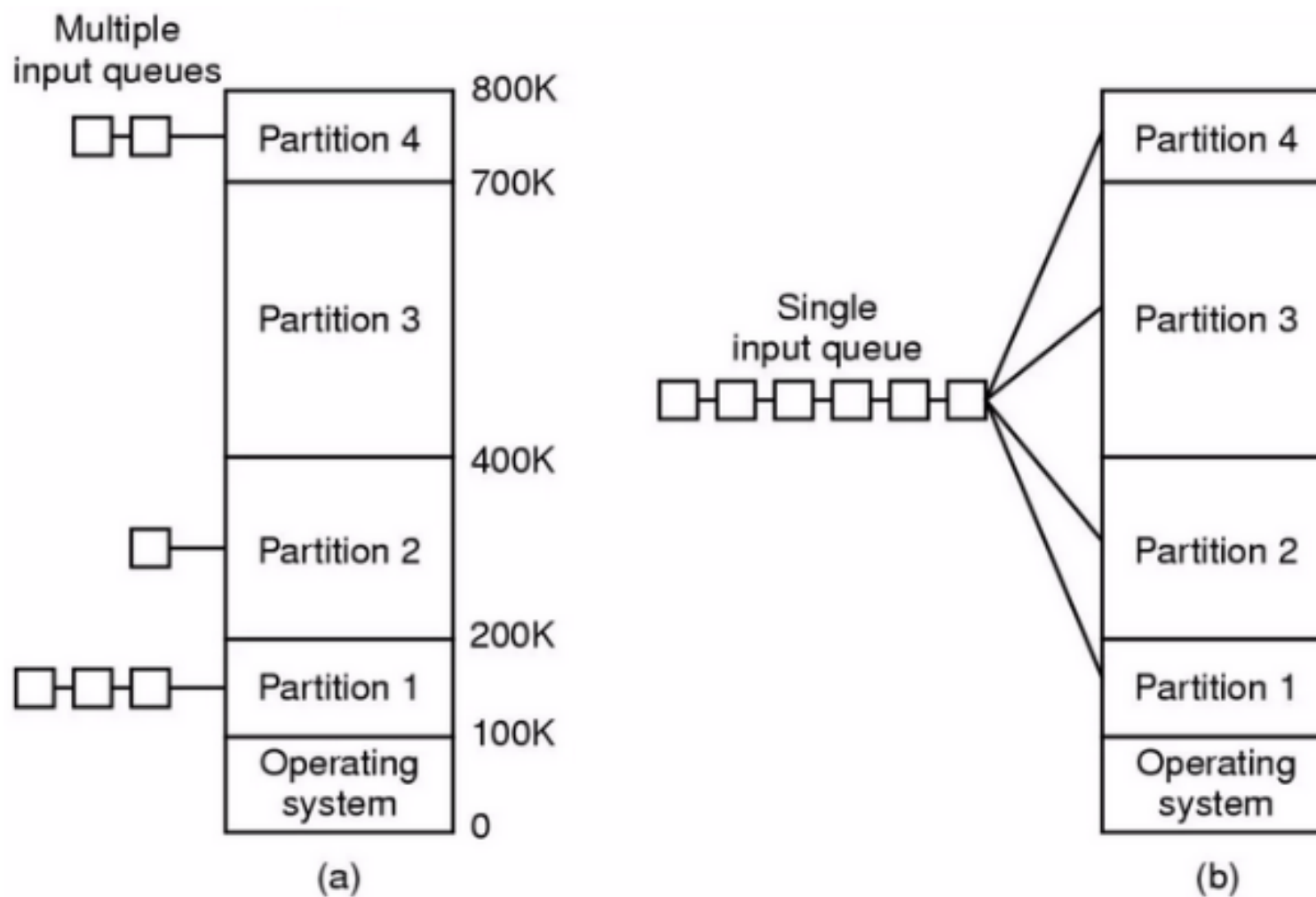
Wieloprogramowość:

Prosty model wykorzystan

- n -procesów, każdy spę
- p^n – prawdopodobieńs
- wykorzystanie CPU =



Wieloprogramowość: wykorzystanie pamięci



Wieloprogramowość ze stałym podziałem pamięci

a.) Osobne kolejki zadań do każdej partycji

- zadanie trafia do kolejki do najmniejszej pasującej partycji
- duże partycje mogą pozostawać puste, nawet gdy są zadania w innych kolejkach, które mogłyby się zmieścić
- małe zadania muszą czekać

b.) Wspólna kolejka do wszystkich segmentów:

- zadanie z początku kolejki trafia do pierwszej wolnej partycji
- może powodować marnowanie pamięci na małe zadania
- potrzebny alg. wyszukiwania zadan w kolejce pasujących do wolnego obszaru - ale to dyskryminuje małe zadania

Wieloprogramowość z dynamicznym podziałem pamięci

- obszary tworzone dynamicznie dopasowane do rozmiaru zadania
- brak fragmentacji wewnętrznej - efektywne wykorzystanie pamięci RAM
- konieczność wykonywania upakowań (defragmentacji) w celu uniknięcia fragmentacji zewnętrznej (niewykorzystane obszary pomiędzy zaalokowanymi blokami), co zwiększa użycie CPU

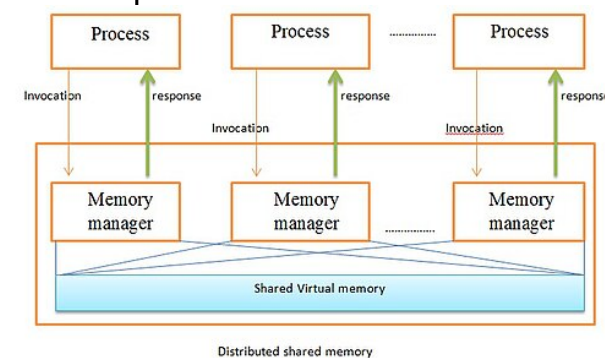
Obecnie stosowane rozwiązanie: pamięć wirtualna, segmentacja o stałym rozmiarze i mechanizm stronicowania

Systemy z podziałem czasu

- praca wsadowa a praca interakcyjna
- wieloprogramowość → **wielozadaniowość**, wykonywanie wielu procesów w określonym przedziale czasu, wymaga mechanizmu podziału czasu
- system z podziałem czasu – wielu użytkowników dzieli jeden komputer, CPU jest przydzielane procesom w małych porcjach czasu (kwantach), gdy kwant się kończy to CPU przedzielane innemu procesowi
- systemy z podziałem czasu są skomplikowane, gdyż wymagają:
 - realizacji mechanizmów działań współbieżnych
 - zarządzania pamięcią
 - ochrony pamięci
 - planowania przydziału CPU
 - administrowania pamięcią dyskową
 - systemu plików dostępnych bezpośrednio

Systemy HPC (*High Performance Computing*)

- **wieloprocusowość** (*multiprocessing*) - systemy posiadające więcej niż jeden procesor pozwalające na równoczesne przetwarzanie zadań
- **SMP** (*Symmetric MultiProcessing*)
 - współdzielona pamięć, I/O
 - wspólny system operacyjny
 - jednostki CPU są równoważne
- **DSM** (*Distributed Shared Memory*)
 - rozdzielone fizycznie pamięci są adresowane wspólną adresową
 - tańszy niż SMP i dobrze się skaluje
 - pozwala na obsługę dużych baz danych
 - wolniejszy dostęp do danych od SMP



Systemy HPC (*High Performance Computing*)

- **MPP** (*Massively Parallel Processor*)
 - setki/tysiące procesorów, rozproszona pamięć
 - każdy CPU posiada własny system operacyjny, własną pamięć
 - podsystemy komunikują łączami o dużej szybkości
- **klastry obliczeniowe**
 - zrównoleglenie przez węzły (liczba węzłów > liczba CPU)
- **konstelacje obliczeniowe**
 - zrównoleglenie przez wielorprocesowość (liczba CPU > liczba węzłów)

Zalety systemów wieloprocessorowych

- podział zasobów
- przyspieszenie obliczeń
- niezawodność
- komunikacja

Rygorystyczne systemy czasu rzeczywistego

- sterowniki urządzeń o ściśle określonym zastosowaniu: nadzorowanie procesów produkcyjnych, eksperymentów naukowych, kierowanie sygnalizacją świetlną, autopilot, itp.
- działanie podlega ostrym rygorom czasowym, nawet kosztem efektywnego wykorzystania systemu

Łagodne systemy czasu rzeczywistego

- działanie podlega złagodzonym rygorom czasowym
- niespełnienie wymagań czasowych nie jest niebezpieczne ale powoduje pogorszenie jakości usług świadczonych przez system
- zastosowania: techniki multimedialne, tworzenie wirtualnej rzeczywistości, urządzenia zdolne do samodzielnej eksploracji

Obsługa urządzeń wejścia-wyjścia

Jak ma współdziałać jednostka centralna z urządzeniami wej-wyj?

- sterownik urządzenia, mapowanie rejestrów urządzenia, MMIO
- aktywne czekanie
- odpytywanie
- przerwania
- bezpośredni dostęp do pamięci

Przerwania

Sterowniki i moduły

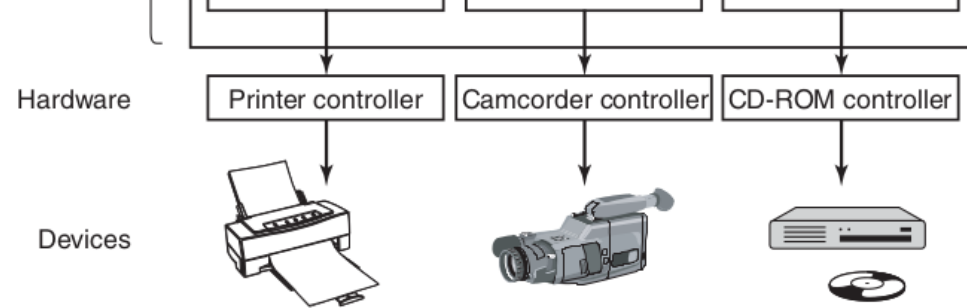


Figure 5-12. Logical positioning of device drivers. In reality all communication between drivers and device controllers goes over the bus.

Sterowniki i moduły sterujące urządzeń

- **Sterownik urządzenia** (*device controller*) związany jest z konkretnym urządzeniem i rozporządza lokalnym buforem i zbiorem rejestrów o specjalnym przeznaczeniu. Odpowiada za przesyłanie danych między urządzeniem zewnętrznym, a własnym buforem.
- **Moduł sterujący/obsługi urządzenia** (*driver*) jest odpowiedzialny od strony systemu operacyjnego za komunikację ze sterownikiem urządzenia.

```
00:1f.3 SMBus: Intel Corporation 8 Series SMBus Controller (rev 04)
Subsystem: Lenovo Device 220c
Flags: medium devsel, IRQ 18
Memory at e0638000 (64-bit, non-prefetchable) [size=256]
I/O ports at efa0 [size=32]
Kernel driver in use: i801_smbus
Kernel modules: i2c_i801
```

slot 00:1f.3 szyna nr 00, urządzenie nr 1f, funkcja nr 3

Zob.: `/proc/iomem`, `/proc/ioports`, `lspci -v`

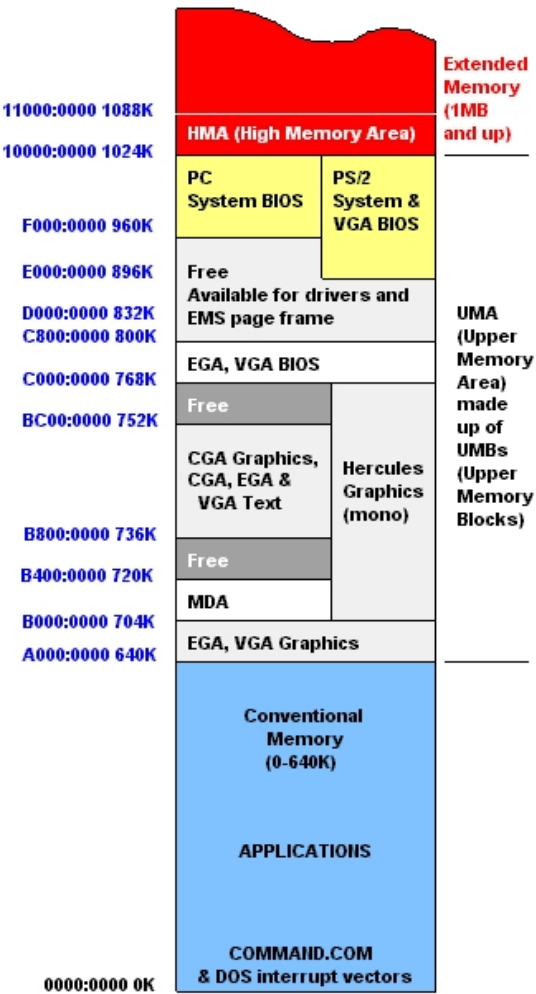
MMIO (memory-mapped input/output)

- komunikacja z urządzeniem - zapis/odczyt rejestrów urządzenia
- zbiór rejestrów urządzenia tworzy **przestrzeń portów wej-wyj**
 - potrzebne specjalne operacje dostępu IN-OUT
- odwzorowanie rejestrów w przestrzeni adresowej pamięci **MMIO** (*memory-mapped input/output*)
 - nie potrzeba dodatkowych instrukcji
 - zapis i odczyt procesora do adresów pamięci, skutkuje zapisami i odczytami do fizycznych rejestrów urządzenia
 - ochrona dostępu realizowana za pomocą mechanizmów ochrony pamięci
 - część przestrzeni adresowej jest zarezerwowana dla urządzeń - w systemach 16 i 32 bitowych z niewielką przestrzenią adresową występuje zjawisko dziury w pamięci¹⁶

¹⁶<https://pl.wikipedia.org/wiki/MMIO>

MMIO (memory-mapped input/output)

From Computer Desktop Encyclopedia
© 2001 The Computer Language Co. Inc.



0.0000 M	0.0039 M	Reserved
0.0039 M	0.6133 M	System RAM
0.6133 M	0.6250 M	Reserved
0.6250 M	0.7500 M	PCI Bus 0000:00
0.7500 M	0.8076 M	Video ROM
0.8125 M	0.8281 M	pnp 00:00
0.8281 M	0.8437 M	pnp 00:00
0.8438 M	0.8594 M	pnp 00:00
0.8594 M	0.8750 M	pnp 00:00
0.8750 M	1.0000 M	Reserved
0.9375 M	1.0000 M	System ROM
1.0000 M	2785.2656 M	System RAM
352.0000 M	360.6260 M	Kernel code
360.6260 M	368.3481 M	Kernel data
370.6484 M	372.5781 M	Kernel bss
2785.2656 M	3000.4531 M	Reserved
3000.4531 M	3000.7656 M	ACPI Non-volatile Storage
3000.7656 M	3020.4648 M	Reserved
3020.4648 M	3022.4961 M	ACPI Non-volatile Storage
3022.4961 M	3022.9961 M	ACPI Tables
...		
4077.2500 M	4077.2969 M	PCI Bus 0000:00
4077.2500 M	4077.2695 M	TPM
4077.2695 M	4077.2969 M	pnp 00:01
4078.0000 M	4078.0039 M	Local APIC
4078.0000 M	4078.0039 M	Reserved
4092.0000 M	4096.0000 M	Reserved

Aktywne czekanie

Aktywne czekanie (*busy waiting*) polega na ciągłym sprawdzaniu stanu zasobów (np. rejestrów, pamięci) w celu ustalenia, czy są one dostępne do dalszej pracy.

1. sprawdź czy drukarka jest gotowa na przyjęcie następnego znaku
2. jeśli nie jest gotowa, to idź do punktu 1
3. jeśli drukarka jest gotowa (po wydrukowaniu znaku), to sprawdź czy jest do wydrukowania nowy znak
4. jeśli jest nowy znak, to idź do punktu 1
5. jeśli nie ma więcej znaków, to drukowanie zostało zakończone

Odpytywanie (*polling*)

Procesor regularnie sprawdza stan kolejnych urządzeń (np. dysków, portów, itp.) w celu ustalenia, czy są one gotowe do wykonania operacji

1. wybierz kolejne urządzenie wymagające obsługi
2. sprawdź, czy to urządzenie wymaga obsługi
3. jeśli tak, to uruchom procedurę obsługi urządzenia
4. jeśli nie, to przejdź do punktu 1.

Obsługa przerw

Przerwanie to żądanie wysyłane do procesora, aby przerwał aktualnie wykonywane zadanie i zajął się obsługą zdarzenia

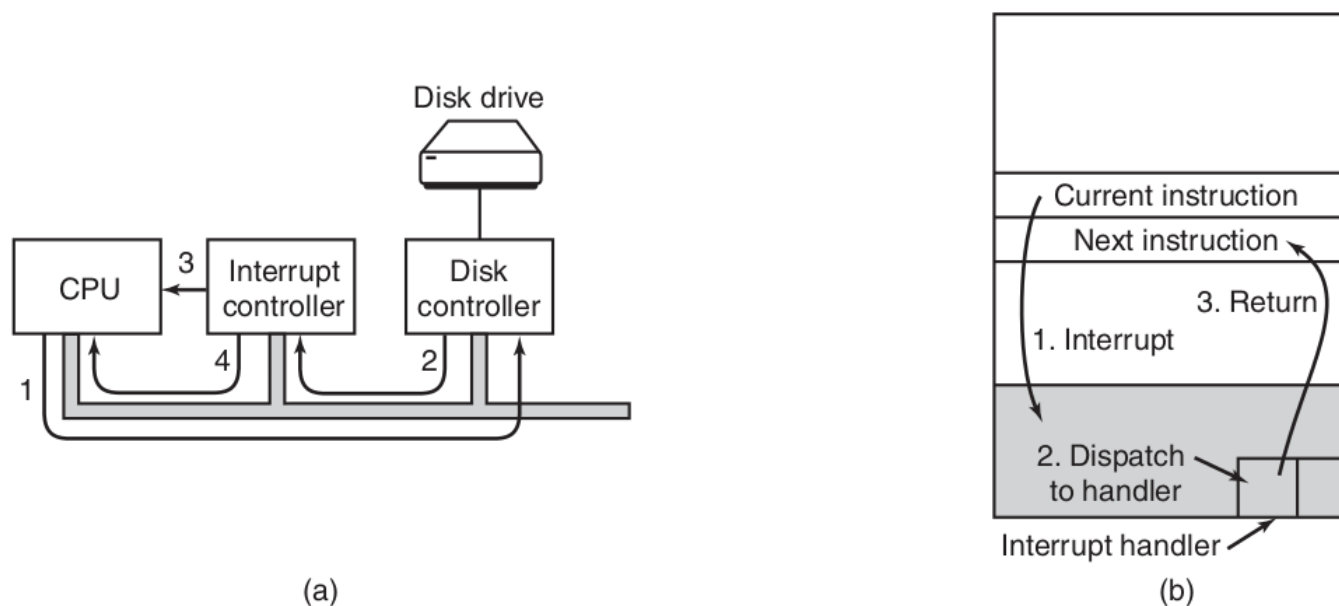


Figure 1-11. (a) The steps in starting an I/O device and getting an interrupt. (b) Interrupt processing involves taking the interrupt, running the interrupt handler, and returning to the user program.

Obsługa przerwań

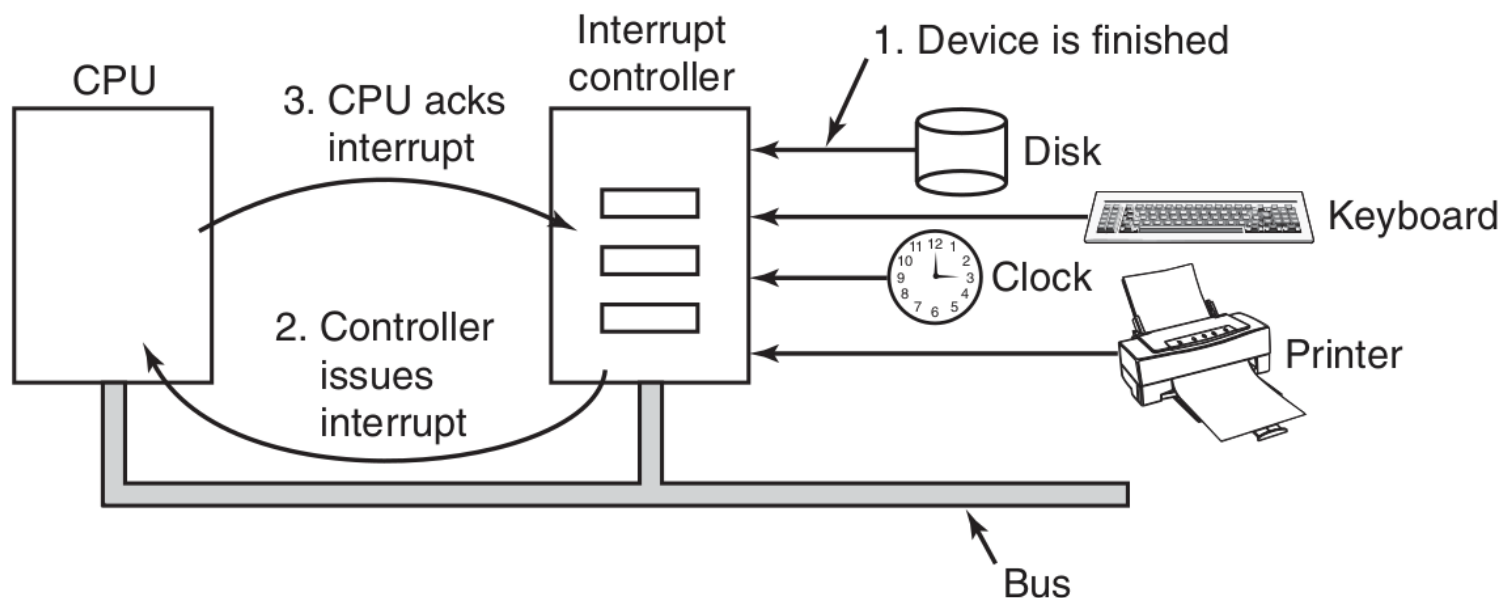


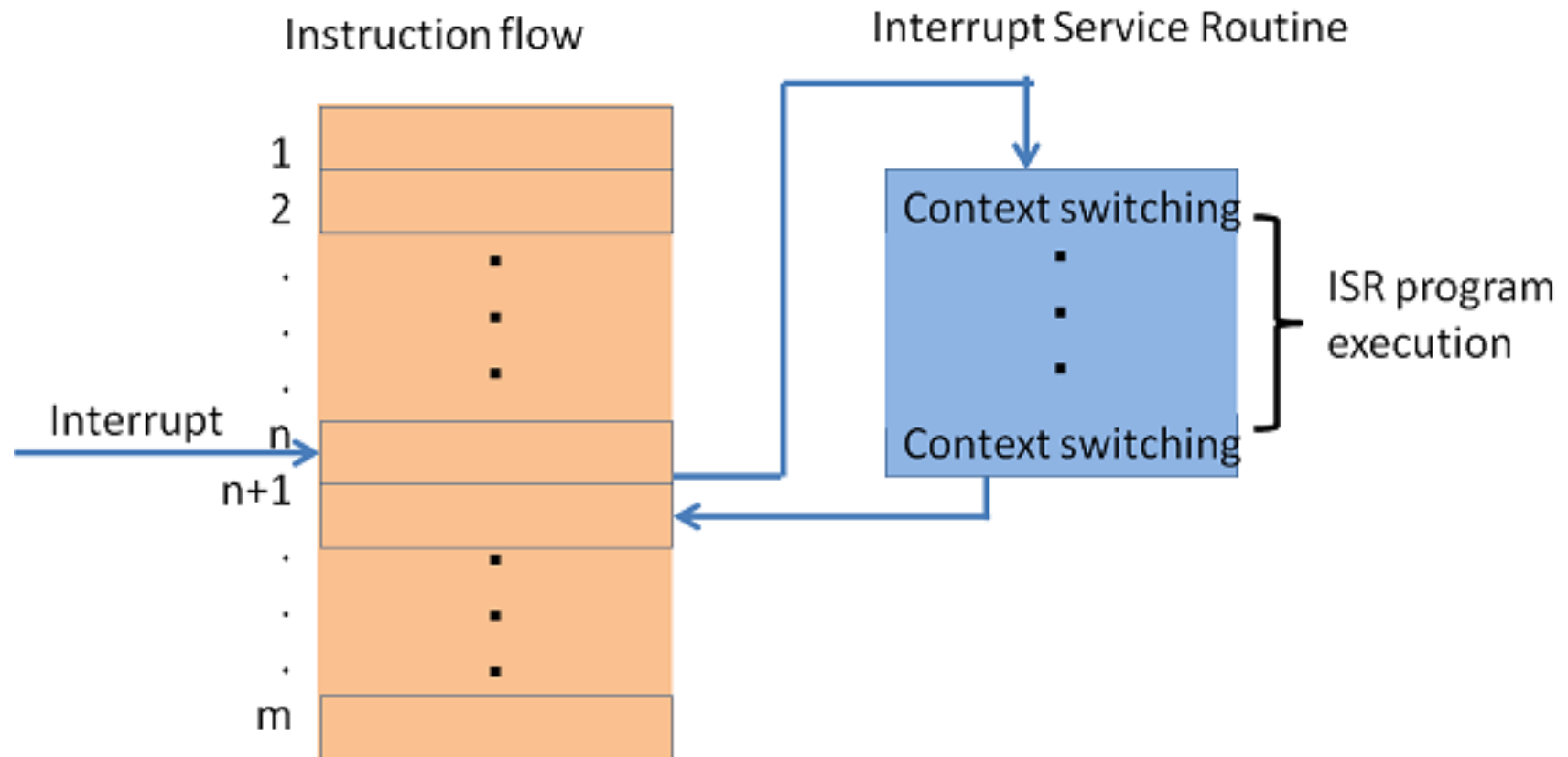
Figure 5-5. How an interrupt happens. The connections between the devices and the controller actually use interrupt lines on the bus rather than dedicated wires.

Przerwania: działanie

- jednostka centralna inicjuje przesyłanie danych przez wprowadzenie pewnych wartości do odpowiednich rejestrów sterownika urządzenia
- sterownik urządzenia rozpoczyna działanie (gromadzenie danych w buforze)
- sterownik urządzenia powiadamia procesor o zakończonej pracy generując określone przerwanie
- procesor wstrzymuje bieżącą pracę, odkłada na stos adres przerwanego rozkazu, określa źródło przerwania i przekazuje sterowanie do procedury obsługi przerwania (*interrupt handler, interrupt service routine (ISR)*) wykorzystując wektor przerwań lub odpytywanie
- przesłanie danych z bufora do programu użytkownika
- wznowienie przerwanej pracy

Procedura obsługi przerwania jest częścią modułu obsługi urządzenia, czyli kodu jądra zarządzającego urządzeniem.

Procedura obsługi przerwania



Przerwania: obsługa

- przerwanie sprzętowe może zostać zainicjowane w dowolnym momencie
- procedura obsługi przerwania powinna zostać uruchomiona jak najszybciej i nie może być przerywana
 - wymagania urządzenia: jak najszybsza obsługa przerwania
 - wymagania systemu operacyjnego: jak najkrótsza obsługa
 - co gdy w przypadku obsługi przerwania pojawi się następne?
- możliwe zagnieżdżanie obsługi wywołań lub odraczanie wykonania
- w Linux obsługa przerwania podzielona na etapy:
 - górna połówka (*top half*) - niezbędne operacje
 - dolna połówka (*bottom half*) - odłożone do późniejszej obsługi

Przerwania: podział

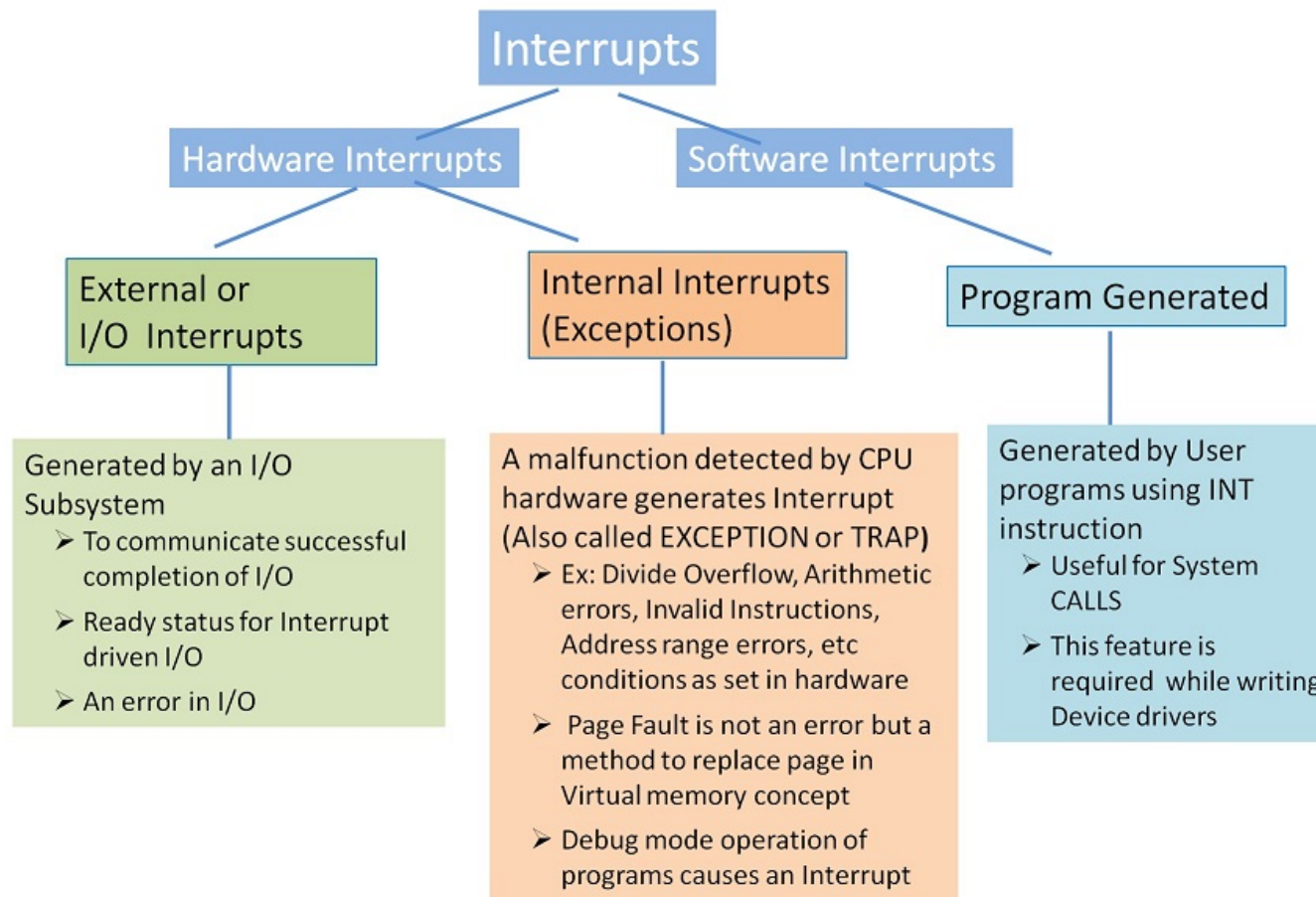
- **asynchroniczne** (przerwania) – generowane przez urządzenia sprzętowe w dowolnym czasie, niezależnie od sygnałów zegarowych procesora, np. przerwanie z karty sieciowej po odebraniu pakietu danych
- **synchroniczne** (wyjątki) – generowane w wyniku wykonania instrukcji, tworzone przez jednostkę sterowania procesora w celu obsłużenia sytuacji wynikającej z wykonania instrukcji, np. dzielenie przez zero, wywołania systemowe

Przerwania: źródła

Systemy z obsługą przerwań są kierowane zdarzeniami generowanymi przez

- przerwania sprzętowe (*interrupts*)
 - zegar - operacje wykonywane w równych odstępach czasu
 - urządzenia I/O - generowane przez sterownik
 - awaria sprzętu
- przerwania wywołane instrukcjami programów
 - błędy programów, nieprawidłowe operacje, załamania (*aborts*)
 - wyjątki (*exceptions*)
 - pułapki (*traps*)
 - przerwania programowe

Przerwania: źródła



Przerwania sprzętowe: podział wg dokumentacji Intelu

- **maskowalne** – mogą być ignorowane
 - dwa stany przerwań: zamaskowany (ignorowany), nie zamaskowany
 - możemy tymczasowo odłożyć uruchomienie procedury obsługi przerwań, do czasu, gdy przerwanie zostanie ponownie włączone
 - wszystkie IRQ (*Interrupt ReQuests*) generowane przez urządzenia I/O powodują powstanie przerwań maskowalnych
 - sygnalizowane przez pin INT procesora.
- **niemaskowalne** – nie mogą być ignorowane.
 - krytyczne zdarzenia (np. awaria sprzętu) zawsze rozpoznawane przez CPU
 - sygnalizowane przez pin NMI (*NonMaskable Interrupt*)

Wyjątki wykrywane przez procesor

Wyjątki generowane w sytuacji wystąpienia nieprawidłowości przy wykonywaniu instrukcji (IEEE 754: niedomiar (*underflow*), nadmiar (*overflow*), dzielenie przez zero, itp.

- **błędy** (*faults*) – instrukcja, która spowodowała błąd może być wznowiona. Register PC (w x86 EIP lub RIP) zapamiętany na stosie zawiera adres instrukcji, która spowodowała wyjątek (np. błąd strony)
- **pułapki** (*traps*) – zgłaszana po wykonaniu instrukcji pułapki, licznik instrukcji PC jest ustawiany na adres kolejnej instrukcji (np. śledzenie programu podczas debugowania)
- **załamania** (*aborts*) – wystąpił poważny błąd i jednostka sterowania nie może zachować adresu instrukcji; zwykle oznacza zakończenie procesu, który spowodował załamanie

Wyjątki programowe – występują na żądanie programisty (np. wywołanie funkcji systemowej, powiadomienie debuggera o zaistnieniu wyjątkowej sytuacji), są obsługiwane za pomocą pułapek

Przerwania: identyfikacja

Każde przerwanie lub wyjątek jest identyfikowane przez liczbę z zakresu od 0 do 255 (Intel nazywa tę 8-bitową liczbę bez znaku **wektorem**).

Tablica deskryptorów przerwania IDT

Interrupt descriptor table

- struktura używana w architekturach x86 do implementacji wektora przerwania
- odwzorowuje numer przerwania na adres funkcji obsługi
- tablica 256 wektorów po 8B, max. rozmiar 2048B
- może rezydować w dowolnym miejscu pamięci, CPU lokalizuje ją za pomocą IDTR

arch/x86/include/asm/irq_vectors.h

0	0..31, system traps and exceptions
1	
	32..127, device interrupts
32	
	int80 syscall interface
128	
129	
	129..255, other interrupts
255	

Tablica deskryptorów przerwań IDT

Rodzaje zdarzeń:

- wyjątki procesora, stałe mapowanie 0-31, niemaskowalne
- przerwania sprzętowe odwzorowują IRQ sprzętu zależnie od kontrolera przerwań 32-127
- przerwania programowe 128-255 definiowane przez BIOS i system operacyjny. Wzbudzane przez instrukcję assemblera INT X przez programy, moduły obsługi urządzeń lub inne funkcje obsługi przerwań.
- w systemie Linux wektor 128 (0x80) wykorzystywany do wywołań funkcji systemowych

Wyjątki procesora x86

#	Wyjątek	Rodzaj	Sygnał
0	<i>Divide error</i>	fault	SIGFPE
1	<i>Debug</i>	fault or trap	SIGTRAP
2	<i>NMI</i>		
3	<i>Breakpoint</i>	trap	SIGTRAP
4	<i>Overflow</i>	trap	SIGSEGV
5	<i>Bounds check</i>	fault	SIGSEGV
6	<i>Invalid opcode</i>	fault	SIGILL
7	<i>Device not available</i>	fault	
8	<i>Double fault</i>	abort	
9	<i>Coprocessor segment overrun</i>	abort	SIGFPE
10	<i>Invalid TSS</i>	fault	SIGSEGV
11	<i>Segment not present</i>	fault	SIGBUS
12	<i>Stack exception</i>	fault	SIGBUS
13	<i>General protection</i>	fault	SIGSEGV
14	<i>Page fault</i>	fault	SIGSEGV
15	zarezerwowana przez Intel'a		
16	<i>Floating point error</i>	fault	SIGBUS
17	<i>Alignment check</i>	fault	SIGSEGV
18	<i>Machine check</i>	abort	
19	<i>SIMD floating point exceptoin</i>	fault	SIGFPE

Wyjątki procesora x86

- jądro systemu dostarcza funkcje obsługi wyjątków (*exception handler*)
- wyjątki 20-31 - zarezerwowane do przyszłych zastosowań
- funkcje obsługi zazwyczaj wysyłają sygnały (w Unix/Linux) do procesu powodującego wystąpienie wyjątku

Przyporządkowanie IRQ do urządzeń I/O

IRQ	INT	urządzenie
0	32	zegar
1	33	klawiatura
2	34	kaskada PIC
3	35	drugi port szeregowy
4	36	pierwszy port szeregowy
5	37	karta dźwiękowa
6	38	stacja dysków
7	39	port równoległy
8	40	zegar systemowy
11	43	interfejs sieciowy
12	44	mysz PS/2
13	45	koprocesor matematyczny
14	46	pierwszy łańcuch sterownik dysków EIDE
15	47	drugi łańcuch sterownika dysków EIDE

- PIC *Programmable interrupt controller* dla jednostek jednoprocessorowych
- chip 8259A obsługuje 8 linii IRQ, w połączeniu kaskadowym (pin 2) do 15 linii IRQ
- PIC połączony do pinu INTR procesora

Sterownik przerwań PIC w systemach Intel PC

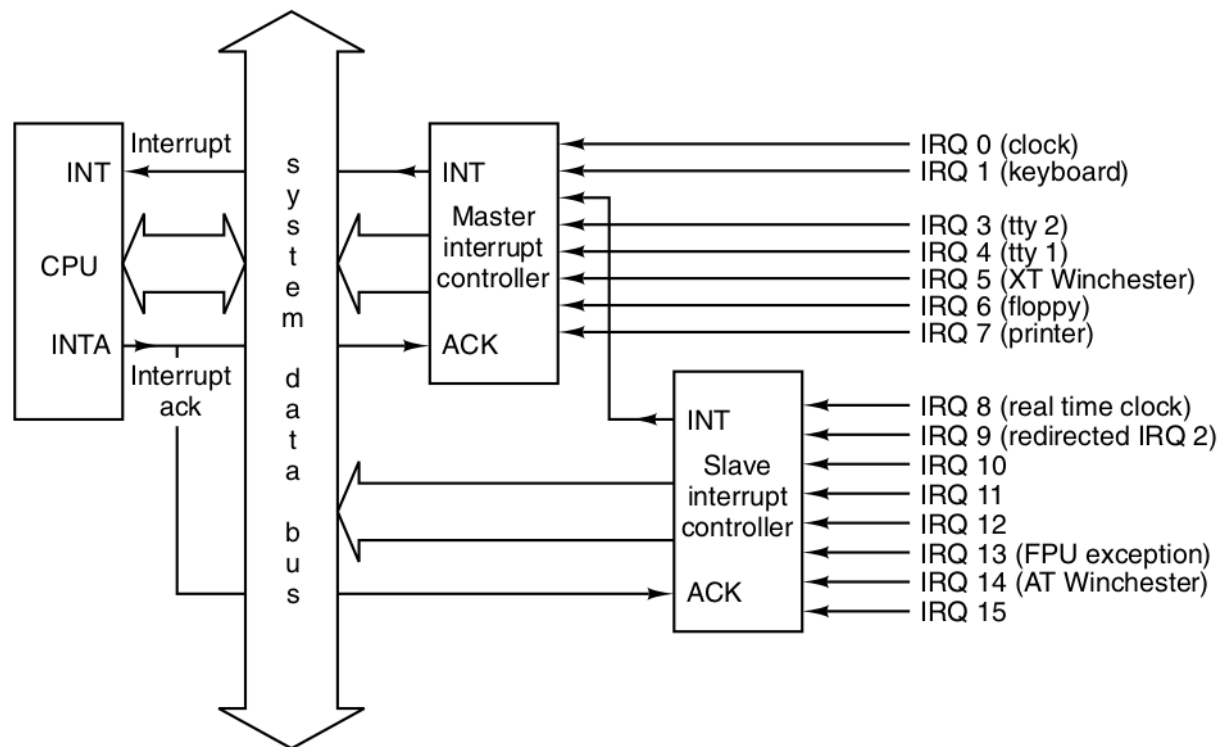


Figure 2-39. Interrupt processing hardware on a 32-bit Intel PC.

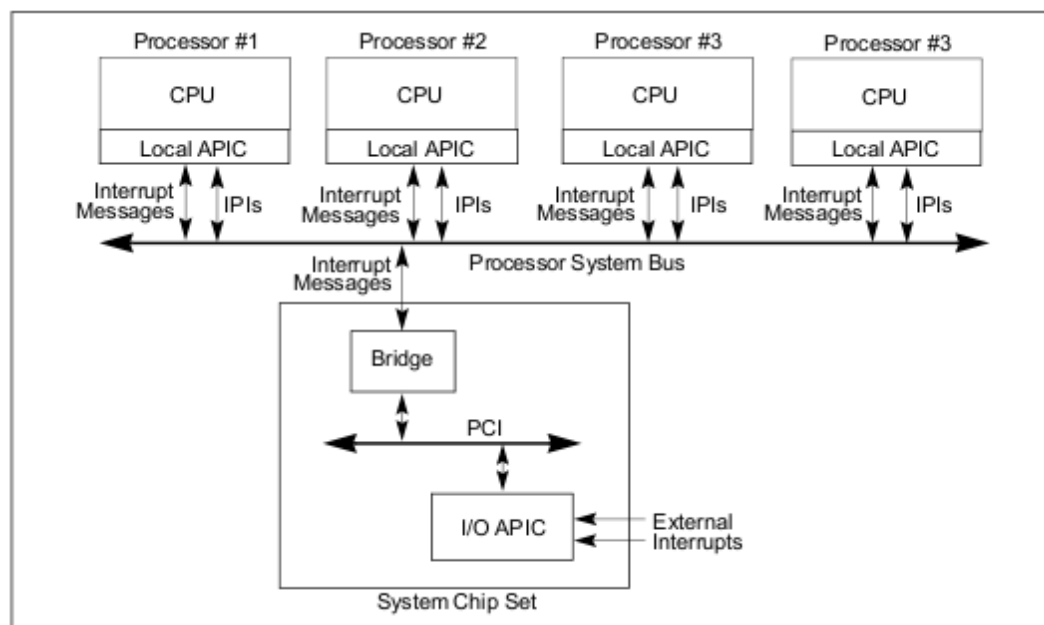
Sterownik przerwań

- linie IRQ łączą urządzenia z pinami wejściowymi programowalnego sterownika przerwań (*Programmable Interrupt Controller*, PIC)
- PIC monitoruje linie IRQ sprawdzając podnoszone sygnały na liniach
- PIC łączy się z pinami INT i MNI procesora i udostępnia mu porty (rejstry) do komunikacji

Gdy urządzenie wymaga obsługi:

1. sterownik urządzenia generuje przerwanie podnosząc sygnał na linii IRQ (*Interrupt ReQuest*, żądanie przerwania)
2. PIC tłumaczy otrzymany IRQ na odpowiedni wektor
3. zachowuje wektor w porcie I/O sterownika przerwań, co pozwala na odczytanie go przez procesor za pomocą szyny danych
4. wysyła sygnał do pinu INT procesora (generuje przerwanie)
5. czeka, aż procesor potwierdzi otrzymanie sygnału przerwania (w tym czasie PIC nie generuje nowych przerwań)
6. CPU potwierdza przerwanie i uruchamia obsługę przerwania

APIC dla procesorów Intel Xeon w systemach SMP



- IPI – *Inter-Processor Interrupt* w systemie wieloprocessorowym (SMP) procesor może zażądać działania ze strony innego procesora (tzw. klepnięcie w ramię): synchronizacja pamięci cache, zatrzymanie systemu, itp.

APIC (*Advanced Programmable Interrupt Controller*)

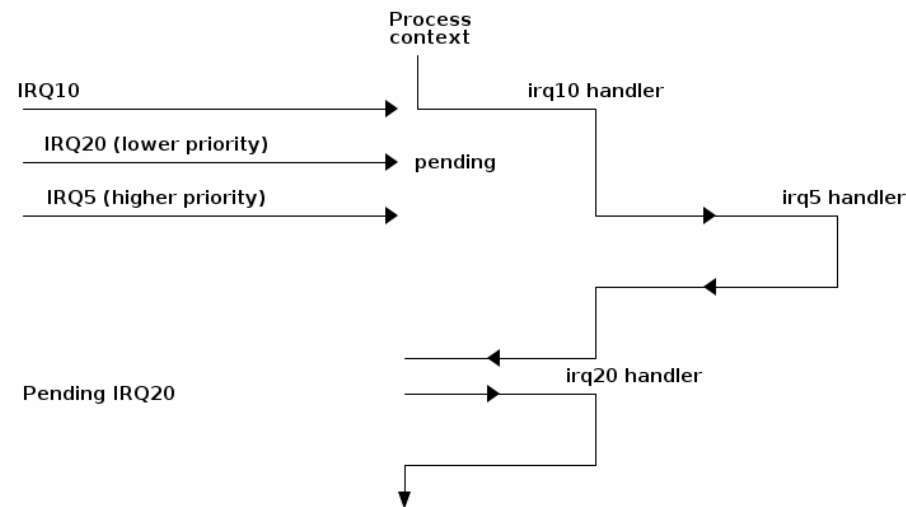
- Local APIC dla każdego CPU
 - obsługa zewnętrznych przerwań dla procesora
 - IPIs - generują i akceptują przerwania między-procesorowe
 - wektor 0-31 wyjątki x86, do 224 wektor z I/O
 - udostępnia wysokiej rozdzielczości zegar (mikrosekundy)
- I/O APIC (opcjonalnie) przy szynie systemowej (po jednym na szynę)
 - przekierowuje przerwania I/O do LAPIC
 - 24 linie IRQ, programowalne rejestry
 - programowalna tabela przekierowań (*Interrupt Redirection Table*):
mapowanie IRQ na wektor przerwań, priorytet, docelowy procesor
 - przerwania zewnętrzne są tłumaczone na komunikaty i wysyłane do LAPIC

Sterowanie przerwaniami

Synchronizacja dostępu do współdzielonych danych pomiędzy procedurą obsługi przerwań i innymi potencjalnymi współbieżnymi działaniami (np. inicjalizacja sterownika, przetwarzanie danych sterownika), często wymaga włączenia i wyłączenia przerwań w kontrolowany sposób:

- z poziomu urządzenia - przez zaprogramowanie rejestrów sterujących urządzeniu
- z poziomu PIC - oprogramowanie PIC włącza/wyłącza daną linię IRQ
 - żądania na wyłączonej linii nie są gubione, PIC prześle je do CPU jak tylko linia zostanie włączona
 - umożliwia szeregowe przetwarzanie przerwań danego typu
- z poziomu procesora - np. instrukcje *cli* (*CLear Interrupt flag*), *sti* (*SeT Interrupt flag*) w architekturze x86 ustawiają maskowanie przerwania, wówczas każde maskowalne przerwanie jest ignorowane (czasowo) przez procesor

Priorytety przerwań



- priorytety przerwań są wspierane przez większość architektur, choć ich obsługa jest problematyczna i rzadko występuje w systemach ogólnego przeznaczenia, przydatne w systemach czasu rzeczywistego
- obsługa przerwania o wyższym priorytecie może zostać wykonana w czasie obsługi przerwania o niższym priorytecie (zagnieżdżenie przerwań)
- przerwania o niższym priorytecie są odłożone do późniejszego wykonania
- wyjątki procesora x86 mają wyższy priorytet niż przerwania I/O

Przerwania: wady

- szybkość transferu wej-wyj jest ograniczona szybkością, z jaką procesor może testować i obsługiwać urządzenie
- procesor jest zajęty zarządzaniem przesyłem danych z wejścia i na wyjście

Szybkie urządzenia wej-wyj oraz urządzenia przesyłające duże ilości danych wymagają bezpośredniego dostępu do pamięci (DMA, *Direct Memory Access*).

Bezpośredni dostęp do pamięci wymaga dodatkowego modułu na magistrali systemowej (moduł DMAC, *DMA Controller*, część mostka południowego).

Przerwania: DMA

Transfer danych wymaga przekazania przez procesor do modułu DMA rozkazu zawierającego informacje:

- czy wymagany jest odczyt/zapis
- adres urządzenia wej-wyj
- adres początkowej komórki pamięci z danymi (na dane)
- liczbę słów do przesłania

Procesor inicjuje przesłanie danych i kontynuuje przetwarzanie do momentu nadejścia przerwania od modułu DMAC.

Sterowanie zdarzeniami (przerwaniemi) rodzi problem synchronizacji w dostępie do zasobów.

Przerwania DMA

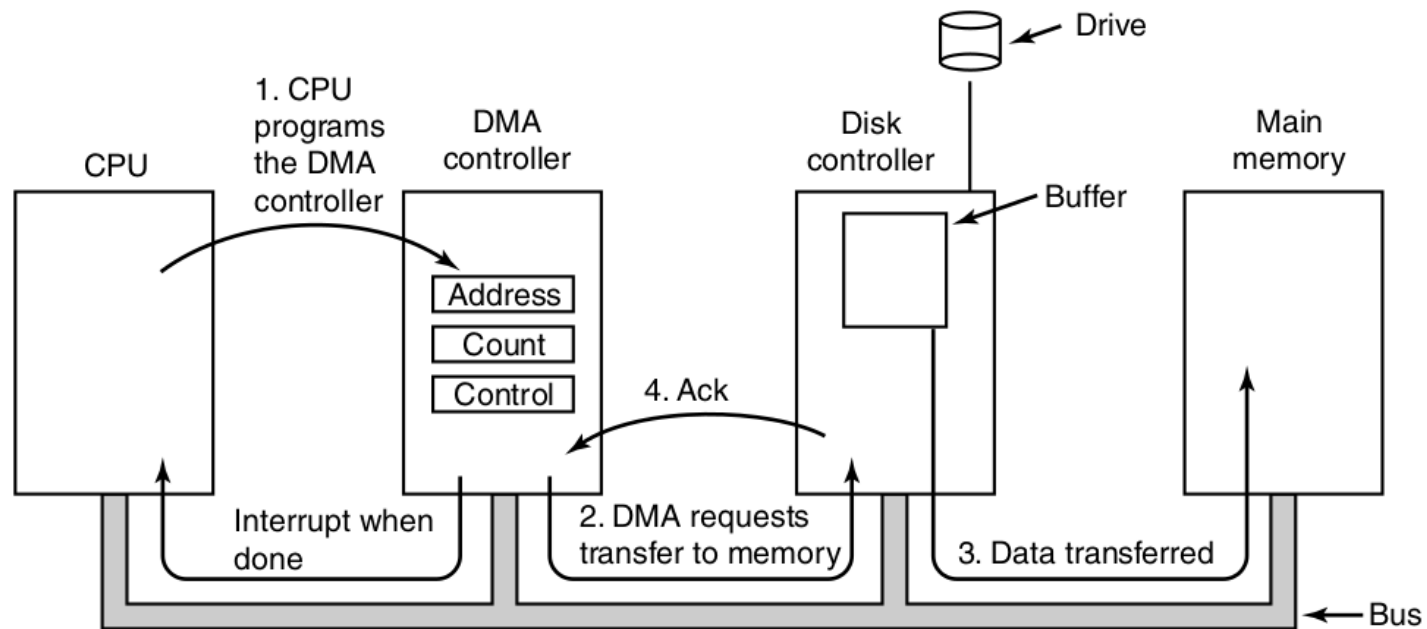


Figure 3-4. Operation of a DMA transfer.

Przerwania sygnalizowane komunikatami (MSI)

Message Signaled Interrupts

- alternatywny system przerwań wprowadzony od PCI 2.2 i PCIe
- obsługiwany przez APIC
- wykorzystuje szynę danych zamiast dedykowanych linii IRQ (PCIe nie posiada linii IRQ)
- komunikacja poprzez zapis do adresów pamięci (MMIO) magistrali PCI (mostka), komunikat nie niesie żadnych danych, jest małym pakietem identyfikującym źródło przerwania
- szybszy w działaniu (rzędy wielkości mniejsze opóźnienia), większa liczba przerwań (dziesiątki przerw na kartę)

Przyporządkowywanie przerwań

```
$ less /var/log/syslog
```

```
Sep 22 09: ... kernel: ttyS00 at 0x03f8 (irq = 4) is a 16550A
Sep 22 09: ... kernel: PCI: Found IRQ 5 for device 00:08.0
Sep 22 09: ... kernel: maestro: Configuring ESS Maestro 2E found at IO 0xD800 IRQ 5
Sep 22 09: ... kernel: parport0: irq 7 detected
Sep 22 09: ... kernel: lp0: using parport0 (polling).
Sep 22 09: ... kernel: ttyS04 at port 0x4880 (irq = 11) is a 16550A
Sep 22 22: ... kernel: eth0: Xircom Cardbus Adapter rev 3 at 0x4800, 00:10:A4:D2:52:55, IRQ 11.
Sep 22 09: ... kernel: ide0 at 0x1f0-0x1f7,0x3f6 on irq 14
Sep 22 09: ... kernel: ide1 at 0x170-0x177,0x376 on irq 15
```

Monitorowanie przerwania sprzętowych

```
# uname -sr
```

```
Linux 2.6.18-92.1.10.el5 x86_64+
```

```
# cat /proc/interrupts
```

	CPU0	CPU1		
0:	799260896	0	IO-APIC-edge	timer
1:	140	100	IO-APIC-edge	i8042
6:	5	0	IO-APIC-edge	floppy
8:	0	0	IO-APIC-edge	rtc
9:	0	0	IO-APIC-level	acpi
12:	115	0	IO-APIC-edge	i8042
15:	7173047	3915	IO-APIC-edge	ide1
169:	0	0	IO-APIC-level	ohci_hcd:usb1, ohci_hcd:usb2
177:	567795	125000	IO-APIC-level	ioc0
185:	6118456	40533	IO-APIC-level	eth0
NMI:	2770	1163		
LOC:	799176280	799176233		
ERR:	0			
MIS:	0			

- edge - zgłoszenie na linii IRQ przez zmianę poziomu (wymaga synchronizacji)
- level - zgłoszenie na linii IRQ przez podniesienie poziomu

Monitorowanie przerwania sprzętowych

```
# uname -sr
```

```
Linux 6.9.3-76060903-generic
```

```
# cat /proc/interrupts
```

	CPU0	CPU1	CPU2	CPU3			
1:	17024	0	0	12	IR-IO-APIC	1-edge	i8042
8:	0	0	0	0	IR-IO-APIC	8-edge	rtc0
9:	47343	2532	0	0	IR-IO-APIC	9-fasteoi	acpi
12:	100109	0	2004	0	IR-IO-APIC	12-edge	i8042
14:	0	0	0	0	IR-IO-APIC	14-fasteoi	INT34BB:00
16:	0	0	0	0	IR-IO-APIC	16-fasteoi	idma64.0, i2c_designware.0, i801_smbus
120:	0	0	0	0	DMAR-MSI	0-edge	dmr0
121:	0	0	0	0	DMAR-MSI	1-edge	dmr1
122:	0	0	0	0	IR-PCI-MSI-0000:00:1c.0	0-edge	PCIe PME
...							
164:	11391	0	0	0	IR-PCI-MSIX-0000:3d:00.0	1-edge	nvme0q1
...							
NMI:	0	0	0	0	Non-maskable interrupts		
LOC:	49586732	34698221	31900639	31100941	Local timer interrupts		
SPU:	0	0	0	0	Spurious interrupts		
PMI:	0	0	0	0	Performance monitoring interrupts		
IWI:	7828372	588193	485311	404572	IRQ work interrupts		
RTR:	0	0	0	0	APIC ICR read retries		
RES:	451370	469078	472718	500172	Rescheduling interrupts		
CAL:	9657155	6803847	6046705	5690671	Function call interrupts		
TLB:	3807589	4242390	4227898	4149771	TLB shootdowns		
TRM:	16	16	16	16	Thermal event interrupts		
THR:	0	0	0	0	Threshold APIC interrupts		
DFR:	0	0	0	0	Deferred Error APIC interrupts		
MCE:	0	0	0	0	Machine check exceptions		
MCP:	356	349	349	349	Machine check polls		
ERR:	0						
MIS:	0						

Przerwania w Linux: górna i dolna połówka

- górna połówka (*top half*)
 - uruchamiany bezpośrednio po odbiorze sygnału przerwania
 - realizuje tylko te czynności, które mają kluczowe znaczenie dla funkcjonowania urządzenia (przyjęcie przerwania, restart urządzenia)
 - uruchomienie funkcji obsługi przerwań urządzeń
 - zlecenie wykonania odroczonej akcji (dolnej połówki)
 - w tym czasie blokowana jest obsługa pozostałych przerwań
 - wykonywanie zbyt długich operacji podczas blokowania przerwań powoduje opóźnienia, problemy z wydajnością i gubienie przerwań
- dolna połówka (*bottom half*)
 - obejmuje czynności obsługi, które mogą być odłożone w czasie i wykonane w momencie mniejszego obciążenia systemu
 - przerwania nie są blokowane w czasie obsługi

Odroczone akcje w Linux

softIRQ

- działają w kontekście przerwania, zlecane przez funkcję obsługi przerwania
- przydzielane statycznie, użycie ograniczone do wybranych podsystemów z wymogami małych opóźnień i wysokiej częstotliwości
- obsługa może być zlecana równolegle na wielu CPU
- zarządzane przez wątki jądra `ksoftirq`

tasklet - szczególny rodzaj softIRQ

- działają w kontekście przerwania
- mogą być dynamicznie alokowane (np. podczas ładowania modułu)
- obsługa szeregową, pojedyncza akcja może być wykonana na jednym procesorze

workqueues

- zadania odroczone wykonywane w kontekście procesu

softirq

```
$ uname -sr
```

```
Linux 6.9.3-76060903-generic
```

```
$ cat /proc/softirq
```

	CPU0	CPU1	CPU2	CPU3
HI:	23555600	3297692	3334936	3477097
TIMER:	8655197	3098823	2571420	2336278
NET_TX:	51	10	8	14
NET_RX:	3089136	33591	32130	78008
BLOCK:	367	472	921	324
IRQ_POLL:	45	0	2	4
TASKLET:	268745	2291	2068	2142
SCHED:	21443427	12885249	11074332	10525505
HRTIMER:	2452	13	10	6
RCU:	7329331	6460572	6142318	6003988

Monitorowanie przerwania sprzętowych: klawiatura

Działanie: naciśnięcie i przytrzymanie pojedynczego klawisza

```
root@nscobie ~$ sar -I 1 1
```

Linux 5.14.16-101.fc33.x86_64 (nscobie)

15.11.2021

_x86_64_

(8 CPU)

09:38:15	INTR	intr/s
09:38:16	1	1,00
09:38:17	1	0,00
09:38:18	1	1,00
09:38:19	1	24,00
09:38:20	1	30,00
09:38:21	1	29,00
09:38:22	1	29,00
09:38:23	1	29,00
09:38:24	1	29,00
09:38:25	1	30,00
09:38:26	1	29,00
09:38:27	1	29,00
09:38:28	1	3,00
^C		
Średnia:	1	20,23

Monitorowanie przerwań sprzętowych: myszka/gładzik

Działanie: szybkie ruchy myszką

```
root@nscobie ~$ sar -I 12 1
```

Linux 5.14.16-101.fc33.x86_64 (nscobie)

15.11.2021

_x86_64_

(8 CPU)

09:45:56	INTR	intr/s
09:45:57	12	0,00
09:45:58	12	71,00
09:45:59	12	469,00
09:46:00	12	471,00
09:46:01	12	471,00
09:46:02	12	469,00
09:46:03	12	472,00
09:46:04	12	469,00
09:46:05	12	473,00
09:46:06	12	349,00
09:46:07	12	356,00
09:46:08	12	472,00
09:46:09	12	102,00
09:46:10	12	0,00
^C		
Średnia:	12	273,02

Monitorowanie przerwain sprzętowych: wlan0

Działanie: ping -i 0.002 -c 1000 192.168.1.1

```
$ sar -I 174 1
```

```
Linux 6.9.3-76060903-generic (kis) 01.11.2024 _x86_64_ (8 CPU)
```

	INTR	intr/s
22:44:25		
22:44:26	174	8,00
22:44:27	174	6,00
22:44:28	174	67,00
22:44:29	174	980,00
22:44:30	174	1845,00
22:44:31	174	674,00
22:44:32	174	16,00
22:44:33	174	6,00
22:44:34	174	24,00
22:44:35	174	11,00
^C		

Średnia:	174	363,70
----------	-----	--------

Wieloprogramowość i ochrona sprzętowa

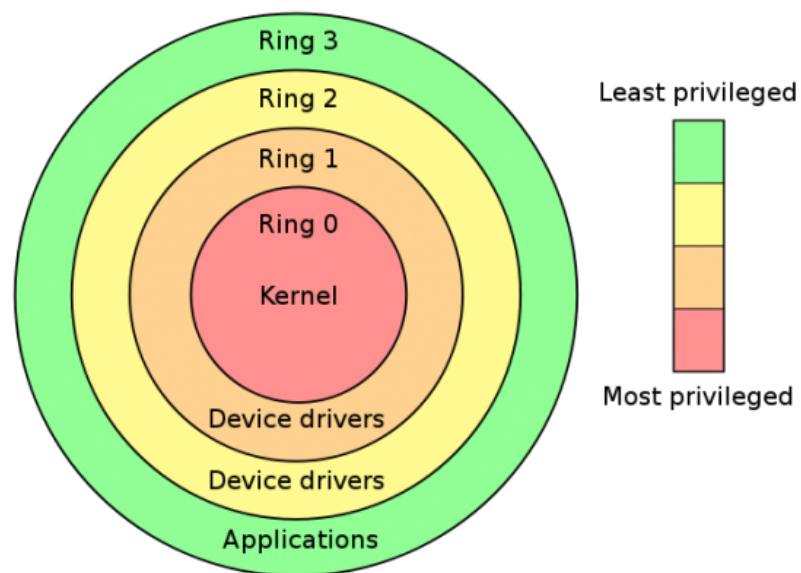
- w systemach wieloprogramowych i z podziałem czasu operacje wej-wyj nakładają się na działanie jednostki centralnej (asynchroniczne wej-wyj).
- zarządzanie wieloma programami wymaga odpowiednich mechanizmów ochrony, aby zapewnić bezpieczeństwo i stabilność systemu operacyjnego
- rolą jądra systemu operacyjnego jest nadzór nad zadobami i udostępnienie interfejsu dostępu do sprzętu
- system operacyjny i jądro systemu są programami, więc potrzebny mechanizm odróżniający kod wykonywany przez użytkownika od kodu jądra systemu

Ochrona sprzętowa

- generowanie przerwania od czasomierza (co kwant czasu) ¹⁷, poprawne działanie jest kluczowe dla stabilności działania systemu
- ochrona pamięci realizowana przez sprzęt (stronicowanie, dawniej adres bazowy i graniczny), zapewnia izolację pamięci między procesami
- ochrona wektora przerwań, procedur wej-wyj, które umożliwiają bezpieczne i kontrolowane przerywanie działania procesora w celu obsługi zdarzeń zewnętrznych
- wszelkie rozkazy wej-wyj są uprzywilejowane, tylko system operacyjny może komunikować się z urządzeniami,
- tryby pracy procesora: tryb użytkownika, tryb jądra (pierścienie uprzywilejowania)

¹⁷W jądrach ≥ 2.6 pojawia się możliwość zastosowania tzw. *dynamic ticks*, tzn. że przerwania zegarowe są generowane w razie potrzeby zarówno wtedy, kiedy system jest zajęty, jak i wtedy kiedy pracuje w trybie "idle" (CPU idle state).

Pierścienie uprzywilejowania



- pierścień 3 - dostępny dla użytkownika
- pierścień 0 - tylko jądro systemu, uprzywilejowane operacje, niedostępny bezpośrednio dla programistów i użytkowników

Dualny tryb pracy

- system operacyjny musi gwarantować, że niepoprawny program nie będzie mógł zakłócić działania innych programów
- niepoprawnie działający program musi generować przerwanie
- ochronie muszą podlegać wszelkie zasoby dzielone
- sprzęt pozwalający odróżnić dwa tryby pracy: **tryb użytkownika** oraz **tryb jądra** (monitora, nadzorcy, systemu, uprzywilejowany)
- każde przerwanie powoduje przejście systemu z trybu użytkownika do trybu jądra
- dualny tryb pracy jest uzupełniony listą uprzywilejowanych rozkazów maszynowych, które mogą być wykonywane tylko w trybie jądra

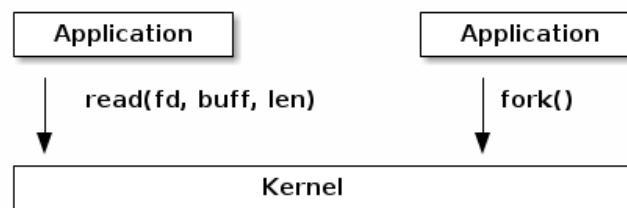
Dualny tryb pracy (cd)

- **tryb jądra (nadzorcy, monitora)** – w tym trybie wykonywane są rozkazy uprzywilejowane, np. rozkazy wejścia-wyjścia, zmieniające stan rejestrów zarządzających pamięcią i czasomierzem, rozkaz *halt*, rozkazy włączania/wyłączania przerwań
- **tryb użytkownika** – aplikacje użytkowe działają z ograniczonymi uprawnieniami, mogą zlecić systemowi wykonanie operacji zastrzezonych (np. dostęp do zasobów) poprzez wywołania systemowe (*system calls*).

Odwołanie do systemu jest traktowane przez sprzęt jak przerwanie programowe. Poprzez wektor przerwań sterowanie przekazywane jest do odpowiedniej procedury obsługi w systemie operacyjnym. .

Przerwania programowe i wywołania systemowe

Z punktu widzenia użytkownika/programisty wywołania systemowe są „usługami” oferowanymi przez jądro systemu aplikacjom użytkownika i przypominają API bibliotek, tj. są opisane jako wywołanie funkcji z nazwą, parametrami i wartością zwracaną¹⁸



Wywołania systemowe nie są w rzeczywistości wywołaniami funkcji, ale konkretnymi instrukcjami asemlera (specyficznymi dla architektury i jądra), które wykonują:

- identyfikują wywołanie systemowe i jego parametry
- przełączają tryb wykonania z trybu użytkownika do trybu jądra
- odbierają rezultat wywołania systemowego

- biblioteki systemowe udostępniają funkcje, które implementują właściwe wywołania systemowe, w ten sposób łatwo z nich korzystać w aplikacjach
- interfejs wywołań systemowych w Linuksie (większości Uniksów) stanowi częściowo biblioteka C (glibc)

Program:	wywołanie funkcji <code>printf()</code>
Biblioteka C:	<code>printf()</code> $\longrightarrow$ <code>write()</code>
Jądro:	wywołanie systemowe <code>write()</code>

- wywołania systemowe są niezbędne, gdyż programy działające w przestrzeni użytkownika nie mogą wykonać kodu jądra
- programy nie mogą bezpośrednio zainicjować wywołania funkcji operacji jądra, ponieważ jądro jest umieszczone i działa w chronionym obszarze pamięci operacyjnej.

Przerwania programowe i wywołania systemowe

- przerwanie programowe realizuje w procesorach rodziny x86 instrukcja `INT $0x80`
- powoduje ona przełączenie systemu do trybu jądra i rozpoczęcie wykonywania wektora przerwania 128, który wskazuje na adres procedury obsługi wywołań systemowych `system_call()` (zdefiniowana w pliku *entry.S*)
- konkretne wywołanie systemowe jest identyfikowane przez numer wywołania, który jest umieszczany w rejestrze `eax`
- rejestry (`ebx`, `ecx`, `edx`, `esi`, `edi`) są (zwykle) używane do przekazywania argumentów wywołania systemowego. Przed użyciem dokonywana jest dokładna weryfikacja przekazanych argumentów.

Przerwania programowe i wywołania systemowe

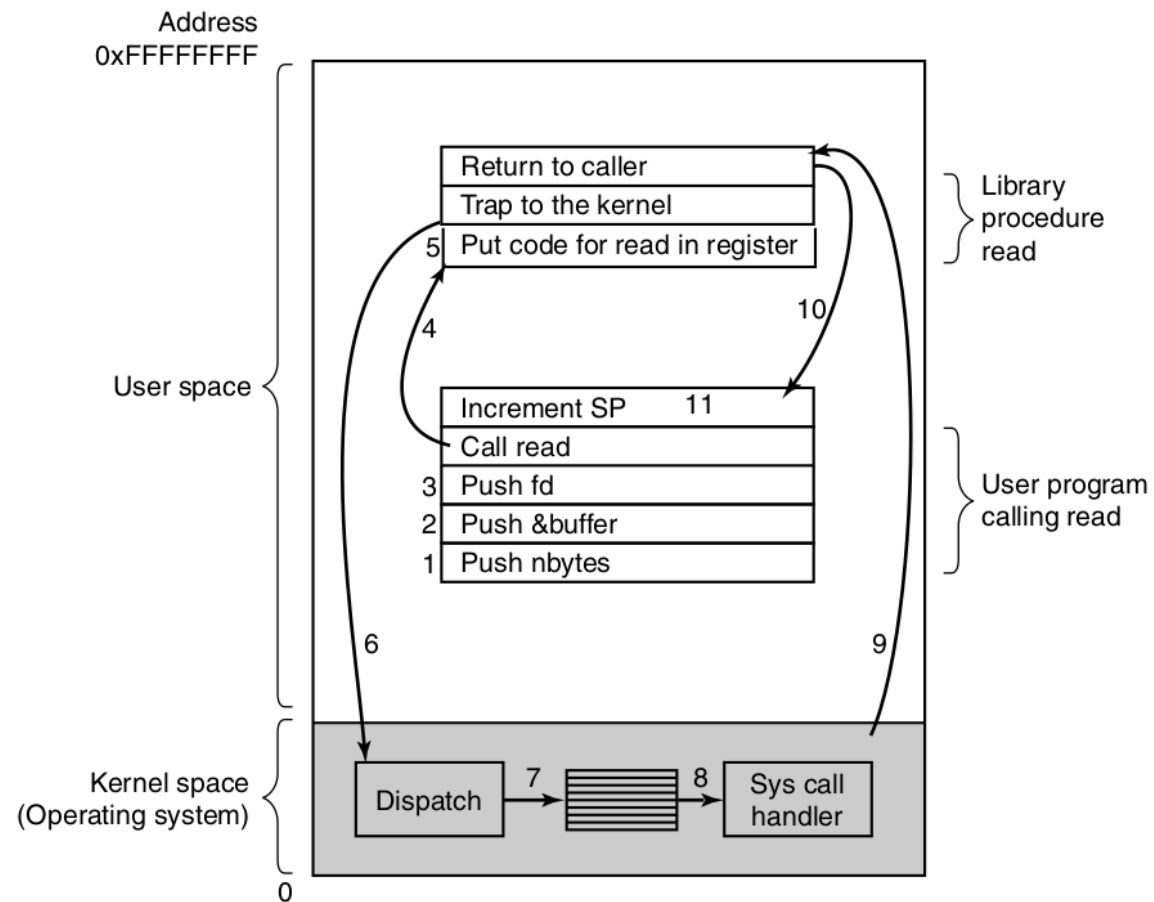


Figure 1-16. The 11 steps in making the system call `read(fd, buffer, nbytes)`.

Przerwania programowe i wywołania systemowe

Fragmenty pliku /usr/include/asm/unistd_64.h

Fedora 24

```
#define __NR_read 0
#define __NR_write 1
#define __NR_open 2
#define __NR_close 3
#define __NR_stat 4
#define __NR_fstat 5
#define __NR_lstat 6
...
#define __NR_mmap 9
#define __NR_mprotect 10
#define __NR_munmap 11
#define __NR_brk 12
...
#define __NR_access 21
#define __NR_pipe 22
#define __NR_select 23
#define __NR_sched_yield 24
...
#define __NR_clone 56
#define __NR_fork 57
#define __NR_vfork 58
#define __NR_execve 59
#define __NR_exit 60
#define __NR_kill 62
#define __NR_uname 63
...
#define __NR_preadv2 327
#define __NR_pwritev2 328
```

Fedora 33

```
...
#define __NR_pkey_free 331
#define __NR_statx 332
#define __NR_io_pgetevents 333
#define __NR_rseq 334
...
#define __NR_pidfd_send_signal 424
#define __NR_pidfd_send_signal 424
#define __NR_io_uring_setup 425
#define __NR_io_uring_enter 426
...
#define __NR_mount_setattr 442
#define __NR_quotactl_fd 443
#define __NR_landlock_create_ruleset 444
#define __NR_landlock_add_rule 445
#define __NR_landlock_restrict_self 446
#define __NR_memfd_secret 447
```


Przerwania programowe: śledzenie

```
$ strace -c cat /etc/hosts > /dev/null
```

% time	seconds	usecs/call	calls	errors	syscall
35.84	0.000944	944	1		execve
11.92	0.000314	28	11	6	openat
11.20	0.000295	29	10		mmap
6.42	0.000169	24	7		close
5.58	0.000147	36	4		brk
5.28	0.000139	23	6		fstat
4.78	0.000126	31	4		mprotect
4.71	0.000124	24	5		read
3.83	0.000101	50	2	1	arch_prctl
3.49	0.000092	46	2		munmap
3.26	0.000086	21	4		pread64
2.01	0.000053	53	1	1	access
0.91	0.000024	24	1		write
0.76	0.000020	20	1		fadvise64
100.00	0.002634	44	59	8	total

Przerwania programowe: śledzenie

```
$ strace -c firefox
```

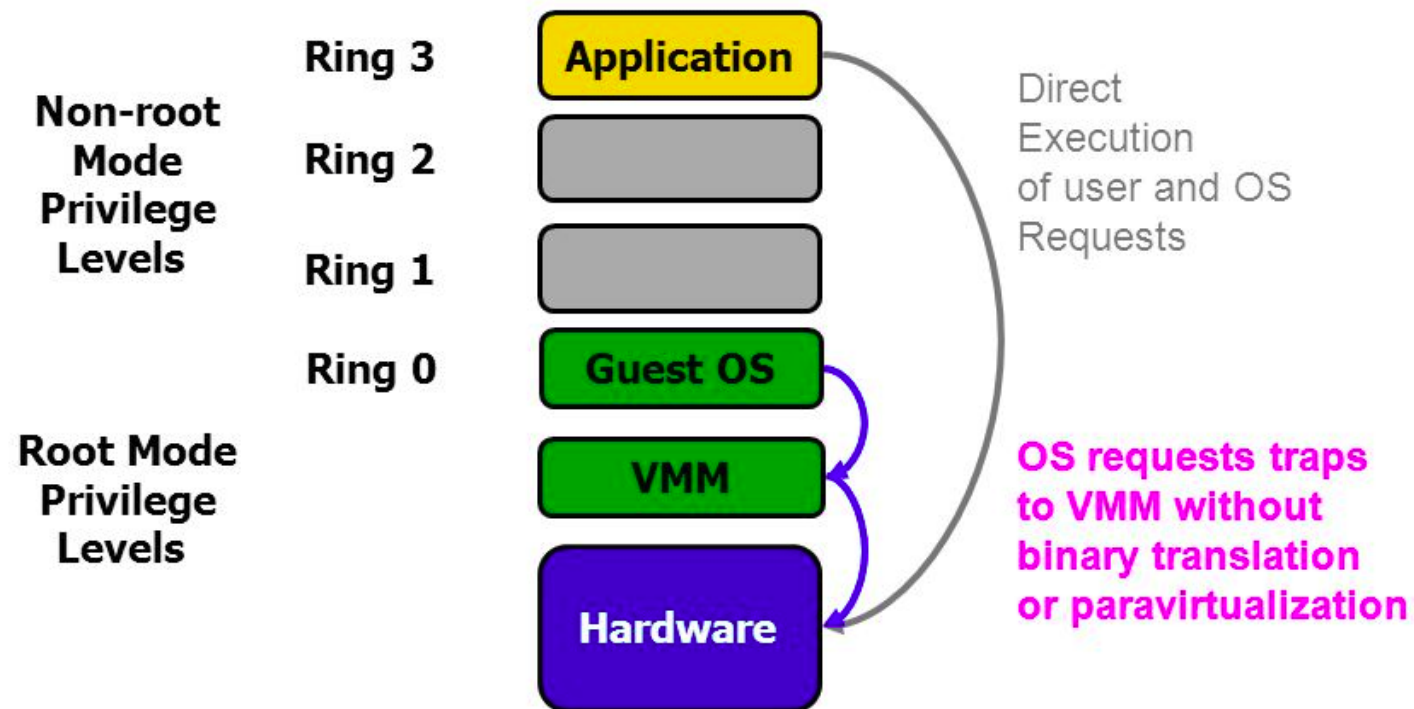
% time	seconds	usecs/call	calls	errors	syscall
26.32	0.364886	45	7963	505	futex
13.59	0.188327	13	14104	12926	recvmsg
11.21	0.155370	14	11038		poll
7.90	0.109478	18	5802		write
7.60	0.105350	18	5705	1	madvise
5.28	0.073184	2152	34	16	wait4
5.19	0.071983	12	5705	3	read
4.38	0.060772	23	2565		mprotect
3.09	0.042794	20	2041		mmap
3.08	0.042744	41	1023		munmap
...					
0.00	0.000000	0	1		set_robust_list
0.00	0.000000	0	1		inotify_init1
0.00	0.000000	0	1		sched_setattr
0.00	0.000000	0	1		sched_getattr
100.00	1.386228	20	68379	14862	total

Czas spędzony w trybie jądra

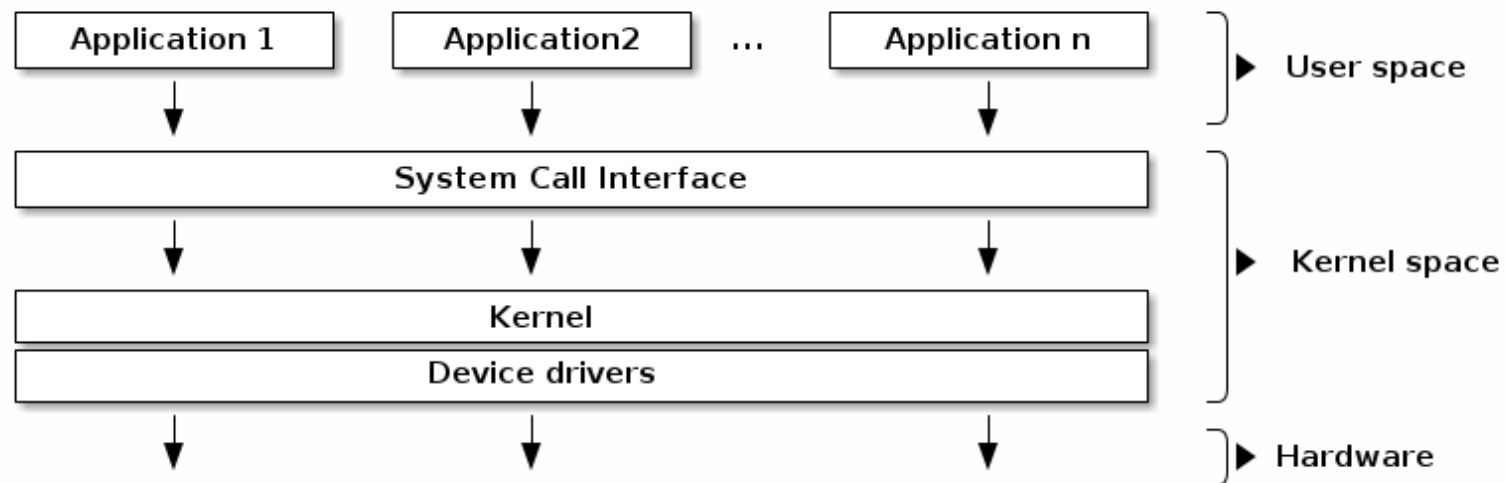
```
$ /bin/time -p ping -i 0.002 -c 500 127.0.0.1 > /dev/null  
real 0.99  
user 0.19  
sys 0.80
```

- real czas rzeczywisty
- user czas CPU spędzony w trybie użytkownika
- sys czas CPU spędzony w trybie jądra

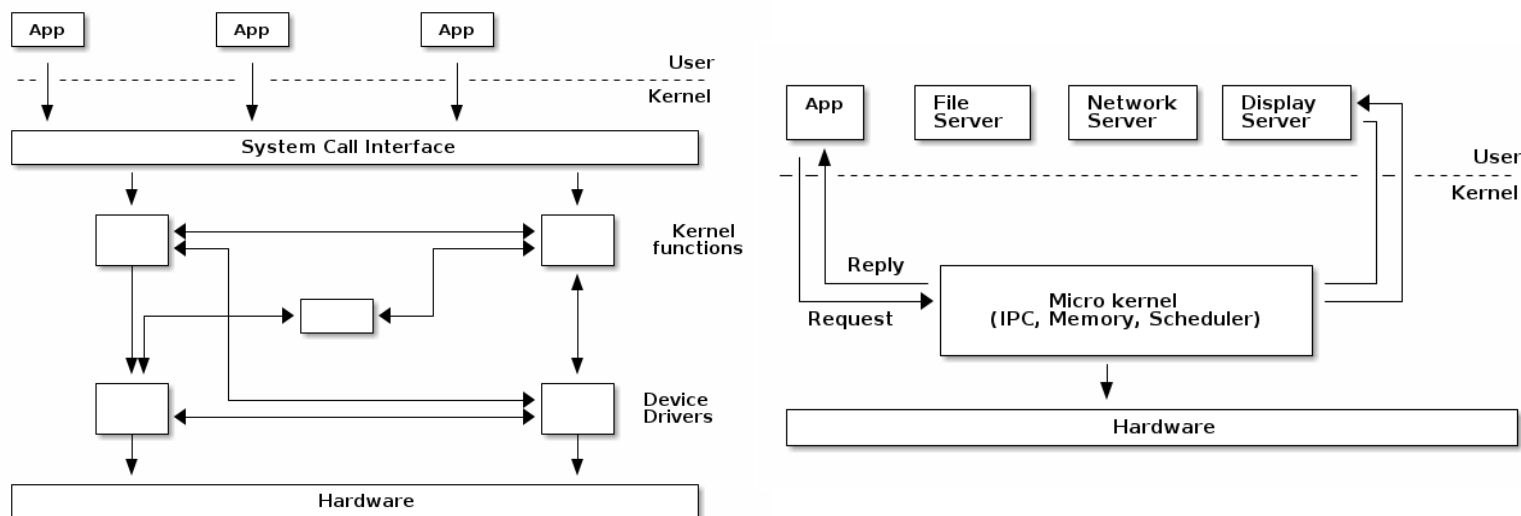
Pierścienie uprzywilejowania i wirtualizacja



Typowa architektura systemu



Monolityczne jądro vs. mikrojądro



- **Jądro monolityczne** nie ma ochrony dostępu pomiędzy różnymi podsystemami jądra i w którym funkcje publiczne mogą być bezpośrednio wywoływane pomiędzy różnymi odsystemami (Linux)
- **Mikrojądro** - duże części jądra są przed sobą chronione i zwykle działają jako usługi w przestrzeni użytkownika. Ponieważ znaczna część jądra działa w trybie użytkownika, pozostały kod działający w trybie jądra jest znacznie mniejszy, stąd termin mikrojądro.
- jądra hybrydowe (Mac OS X, Windows) - jednak większość monolitycznych usług działa tu w trybie jądra, więc można kwalifikować je jako monolityczne
- Debata o wyższości mikrojądra nad monolitycznym jądrem [Tanenbaum–Torvalds debate](#)

Proces vs. program

Proces to program, który jest wykonywany

- jest dynamiczny, jest uruchomioną instancją programu
- jest aktywny, wykonuje się w pamięci zużywając zasoby systemowe
- posiada bieżący stan wykonywania, w tym licznik programu, stos, rejestry i inne zasoby systemowe; licznik programu wskazuje następną instrukcję do wykonania
- wykonanie procesu musi przebiegać w sposób sekwencyjny

Program to zestaw instrukcji, które mają na celu wykonanie określonego zadania

- jest statyczny, to plik (zbiór bitów) przechowywany na dysku
- jest bierny, nie wykonuje się
- nie jest procesem

Procesy: zadania systemu operacyjnego

System operacyjny odpowiada za wykonywanie następujących czynności:

- tworzenie i usuwanie procesów
- zarządzanie stanem procesów (wstrzymywanie i wznowianie procesów)
- dostarczanie mechanizmów komunikacji między procesami
- synchronizacja procesów, zapobieganie problemom takim jak zakleszczenia i wyścigi
- obsługa przerwań i sygnałów
- zapewnienie ochrony i bezpieczeństwa, izolacja procesów i zarządzanie uprawnieniami dostępu do zasobów
- monitorowanie działania procesów i dostarczanie narzędzi do diagnostyki
- zarządzanie przydziałem pamięci dla procesów, wirtualizacja pamięci i ochrona dostępu

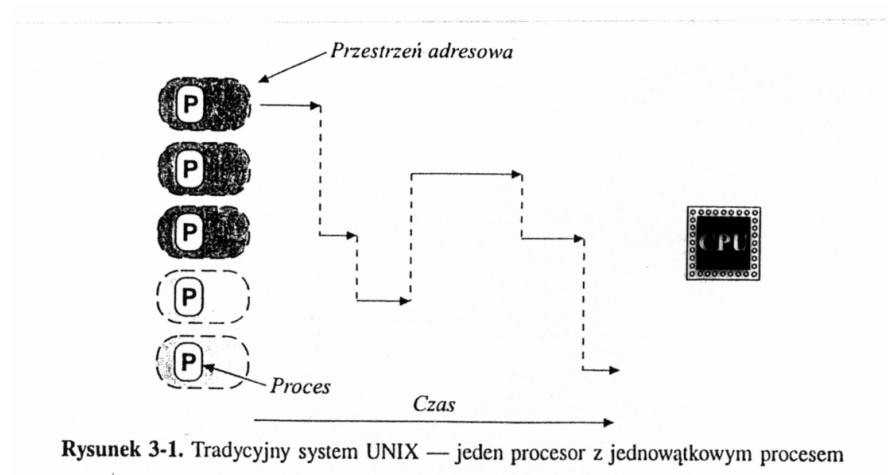
Procesy: rodzaje

Proces stanowi jednostkę pracy w systemie.

System składa się ze zbioru procesów:

- procesy systemu operacyjnego (wykonują kod systemu)
- procesy użytkowników (wykonują kod programów użytkowników)
- procesy działające w tle (demony) - wykonują zadania bez bezpośredniej interakcji z użytkownikiem (usługi)
- procesy interaktywne (np. terminale, aplikacje GUI)
- procesy wsadowe (batch) - uruchamiane bez interakcji użytkownika, często jako zadania wsadowe wykonywane sekwencyjnie

Współbieżność (pseudoparalelizm) - w jenoprocesorowym systemie z podziałem czasu w każdej chwili wykonuje się tylko jeden proces, ale z uwagi na przełączanie kontekstu powstaje wrażenie równoczesnej pracy wielu procesów.



Przetwarzanie równoległe - jednoczesne wykonywanie wielu procesów na różnych procesorach, procesy są rzeczywiście wykonywane jednocześnie.

Tworzenie procesów

Zdarzenia powodujące powstawanie procesów

- inicjalizacja systemu
- wykonanie przez działający proces funkcji systemowej tworzącej procesy
 - Unix/Linux: `fork()`
 - Windows WinAPI: `CreateProcess()`
- żądanie użytkownika w systemie interaktywnym
- inicjacja zadania wsadowego w systemach wsadowych

Zakończenie procesu

Zdarzenia powodujące zakończenie działania procesów:

- normalne wyjście po zakończeniu pracy (dobrowolne)
 - Unix/Linux: `exit()`
 - Windows WinAPI: `ExitProcess()`
- wyjście zakończone błędem (dobrowolne), gdy proces wykryje sytuację uniemożliwiającą dalsze wykonie
- krytyczny błąd (niedobrowolne), w wyniku wykonania niepoprawnej instrukcji
- zabicie przez inny (uprawniony) proces (niedobrowolne), poprzez wywołanie funkcji systemowej
 - Unix/Linux: `kill()`
 - Windows WinAPI: `TerminateProcess()`

Hierarchia procesów

- więź między procesami **potomnymi** (*child*) i **nadrzędnymi** (*parent*) tworzy hierarchię (np. Linux), strukturę drzewiastą. Pierwszy proces (korzeń) startuje podczas inicjacji systemu, np. `init`, `systemd`
- każdy proces posiada tylko jednego rodzica
- proces wraz ze wszystkimi potomkami i przodkami tworzy **grupę procesów**, które wspólnie mogą odbierać sygnały, choć każdy z nich może dostarczać indywidualną funkcję obsługi sygnału lub go zignorować
- nie wszystkie systemy stosują hierarchę.
W Windows procesy nie tworzą wyraźnej hierarchii. Proces nadrzędny uzyskuje **uchwyt** do potomka, za pomocą którego może go kontrolować ale uchwyt może być przekazany innemu procesowi

Hierarchia procesów

```
$ pstree
```

```
systemd-+-ModemManager---2*[{ModemManager}]
        |-NetworkManager---2*[{NetworkManager}]
        |-accounts-daemon---2*[{accounts-daemon}]
        |-acpid
        |-atop
        |-atopacctd
        |-avahi-daemon---avahi-daemon
        |-blueman-mechani---2*[{blueman-mechani}]
        |-bluetoothd
        |-cron
        |-gnome-keyring-d-+-ssh-agent
        |-python3
        |-sshd
        |-systemd-+-(sd-pam)
        |   '-gnome-terminal--+-2*[zsh]
        |                       |-zsh---pstree
        |                       '-3*[{gnome-terminal-}]
        '-systemd-journal
```

```
$ pstree -s $$
```

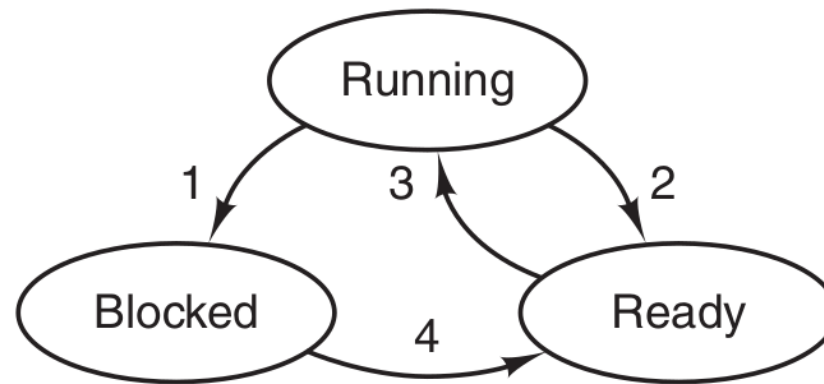
```
systemd---systemd---gnome-terminal---zsh---pstree
```

Procesy: stany

- **gotowy** – proces czeka na przydział procesora
- **bieżący** (aktywny) – aktualnie używający CPU, są wykonywane instrukcje
- **oczekujący** (zablokowany) – czeka na wystąpienie jakiegoś zdarzenia (np. zakończenia operacji wejścia-wyjścia)



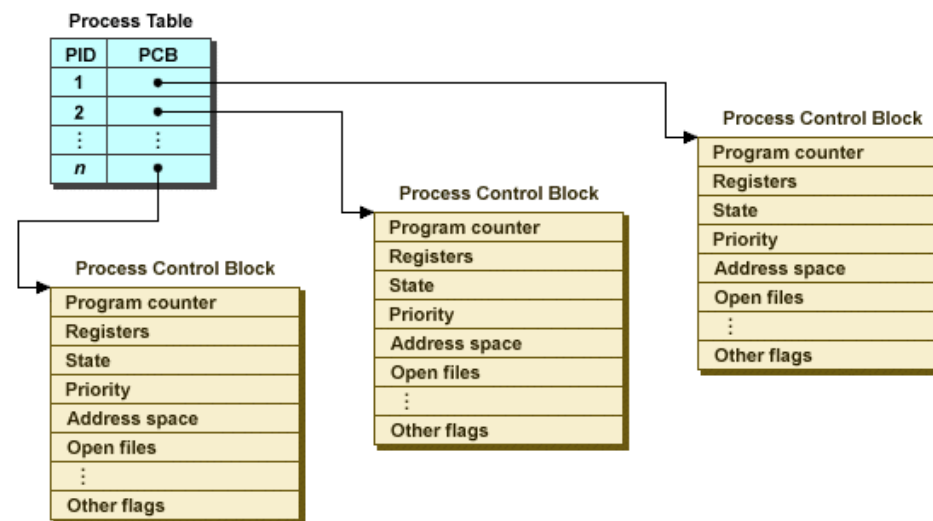
Procesy: zmiana stanu



1. proces blokowany w oczekiwaniu zdarzenie (oczekiwanie na wejście)
2. planista wybiera inny proces do wykonania
3. planista wybiera ten proces do wykonania
4. zachodzi zdarzenie odblokowujące proces (wejście staje się dostępne)

Procesy: zarządzanie

Tablica procesów - zarządzana przez system operacyjny tablica zawierająca **bloki kontrolne procesów**, po jednej strukturze na proces



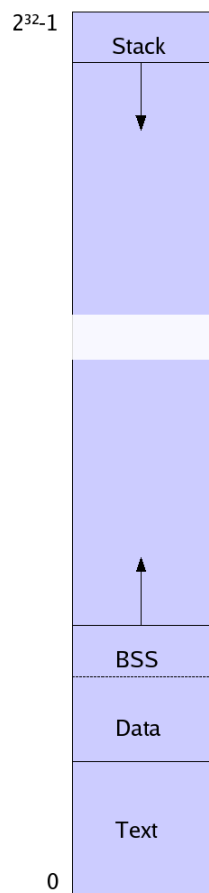
Blok kontrolny procesu zawiera wszystkie niezbędne informacje wymagane do zarządzania procesem, wszystko co musi być zapisane, gdy proces opuszcza stan bieżący, tak aby możliwe było wznowienie jego wykonania

- stan procesu
- numer procesu
- licznik rozkazów, stosu
- rejestry
- ograniczenia pamięci
- wykaz otwartych plików
- informacja o planowaniu przydziału procesora
- informacja o wykorzystanych zasobach (rozliczanie)

Typowe pola bloku kontrolnego procesu

zarządzanie procesem	zarządzanie pamięcią	zarządzanie plikami
rejstry	wskaznik do informacji segmentu tekstu	katalog główny
licznik programu	wskaznik do informacji segmentu danych	katalog roboczy
słowo stanu programu	wskaznik do informacji segmentu stosu	deskryptowy plików
wskaznik stosu		identyfikator użytkownika
stan procesu		identyfikator grupy
priorytet		
parametry szeregowania		
identyfikator procesu		
identyfikator rodzica		
grupa procesów		
sygnały		
czas rozpoczęcia procesu		
wykorzystany czas CPU		
czas CPU procesów potomnych		

Procesy: przestrzeń adresowa



Z procesem związana jest określona wirtualna przestrzeń adresowa, segment programu (*text*), danych zainicjowanych (*data*), danych niezainicjowanych (*bss*), sterty (*stack*), stosu (*heap*).

```
$ size /bin/ps
```

text	data	bss	dec	hex	filename
71928	769	131967	204664	31f78	/bin/ps

Plik: /proc/PID/maps

```
$ pmap PID
```

Przykład: polecenie ps

```
# ps -elf
```

F	S	UID	PID	PPID	C	PRI	NI	ADDR	SZ	WCHAN	STIME	TTY	TIME	CMD
4	S	root	1	0	0	80	0	- 55324		ep_poll	Nov10	?	00:00:15	/usr/lib/systemd/systemd
1	S	root	2	0	0	80	0	-	0	kthrea	Nov10	?	00:00:00	[kthreadd]
1	S	root	4	2	0	60	-20	-	0	worker	Nov10	?	00:00:00	[kworker/0:0H]
1	S	root	6	2	0	60	-20	-	0	rescue	Nov10	?	00:00:00	[mm_percpu_wq]
...														
4	S	avahi	820	1	0	80	0	- 12035		poll_s	Nov10	?	00:00:12	avahi-daemon: running [sc
4	S	root	822	1	0	80	0	- 98305		poll_s	Nov10	?	00:00:03	/usr/libexec/accounts-dae
4	S	rtkit	823	1	0	81	1	- 45995		poll_s	Nov10	?	00:00:00	/usr/libexec/rtkit-daemon
4	S	root	824	1	0	80	0	- 6329		hrttime	Nov10	?	00:00:00	/usr/sbin/smartd -n -q ne
...														
0	S	jkob	1987	1842	0	80	0	- 30643		poll_s	Nov10	pts/2	00:00:00	/bin/bash
...														
4	R	root	32476	25560	0	80	0	- 36787	-		15:42	pts/22	00:00:00	ps -elf

Wątki

Wątek – (lekki) proces działający w tej samej wirtualnej przestrzeni adresowej, co tworzący go (ciężki) proces. Stan wątku jest zdefiniowany przez małą, odrębną ilość danych (własny stan rejestrów i stos).

Grupa równoprawnych wątków współdzieli zasoby procesu

- kod
- przestrzeń adresową
- otwarte pliki
- zasoby systemu
- należy do tego samego użytkownika

Wątki ze sobą współpracują, a nie współzawodniczą (tak jak procesy).

Wątek, to podstawowa jednostka wykorzystania procesora.

Procesy i wątki

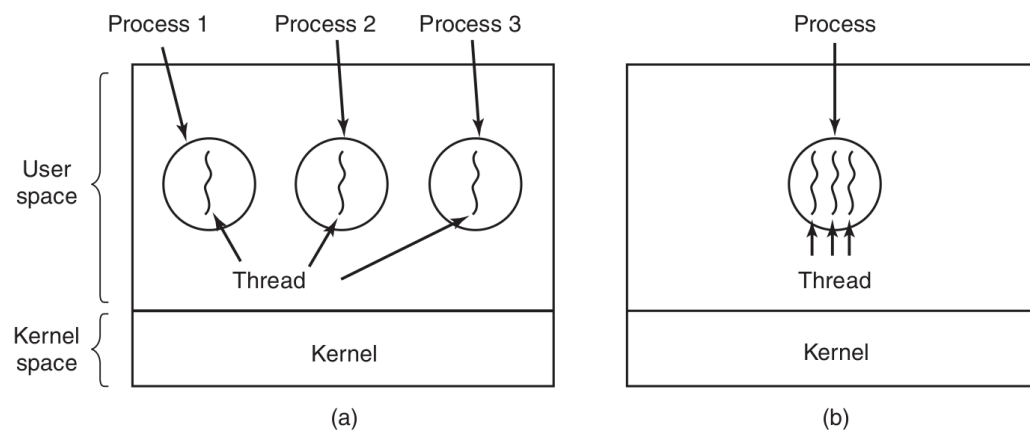
Procesy grupują zasoby, zaś wątki są jednostkami wykonawczymi pracującymi na CPU.

Wątki posiadają własny:

- licznik programu, wskazujący następną instrukcję do wykonania
- stos z historię wywołań procedur
- rejestry, przechowujące wartości zmiennych
- stan, identycznie do stanu procesu: gotowy, bieżący, zablokowany, zakończony

Wielowątkowość - gdy możliwe jest występowanie wielu wątków w procesie

Procesy i wątki



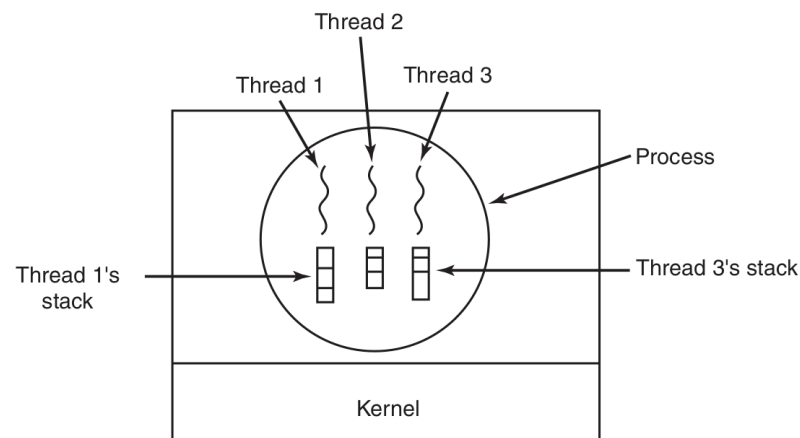
Proces

przestrzeń adresowa
zmienne globalne
otwarte pliki
procesy potomne
zaległe alarmy
sygnały i procedury obsługi sygnałów
informacje dotyczące statystyk

Wątek

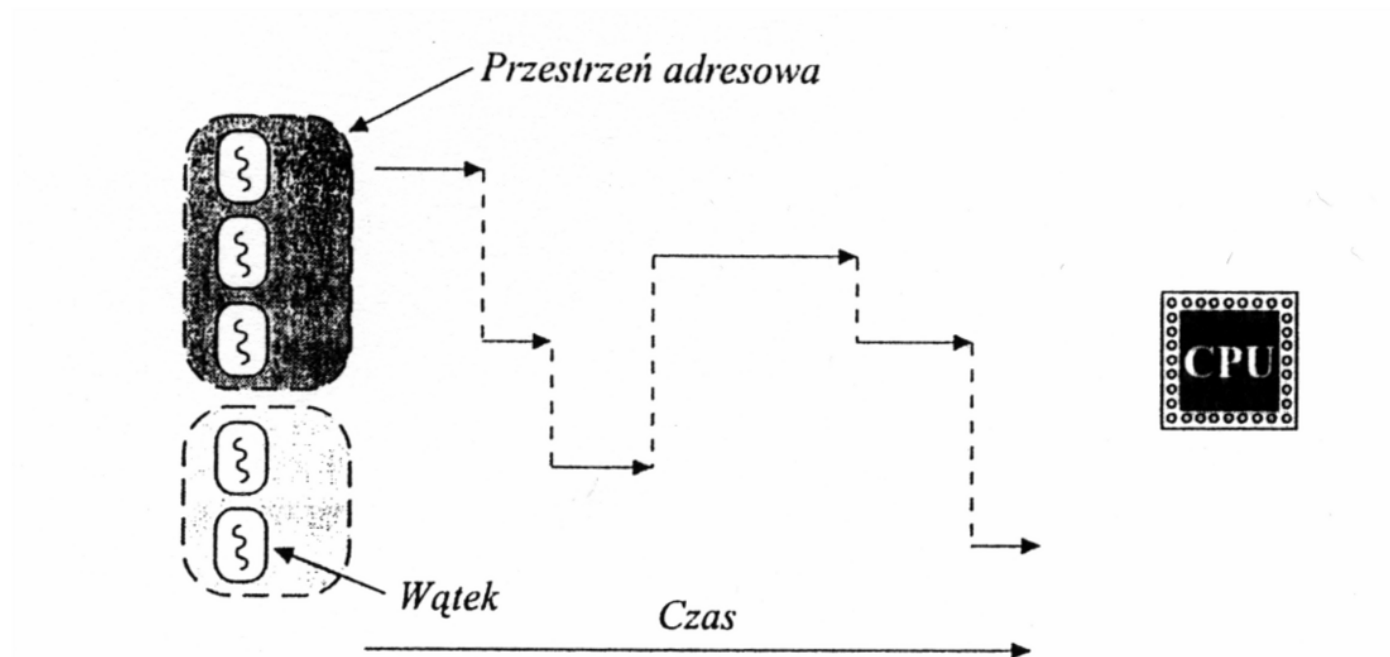
licznik programu
rejstry
stos
stan

Stos procesu



- każdy wątek posiada własny stos
- stos zawiera po jednej ramce dla każdej procedury, która została uruchomiana a z której jeszcze nie nastąpił powrót
- ramka zawiera zmienne lokalne procedury i adres powrotu
- każdy wątek może uruchamiać różne procedury i posiadać odmienną historię wywołań

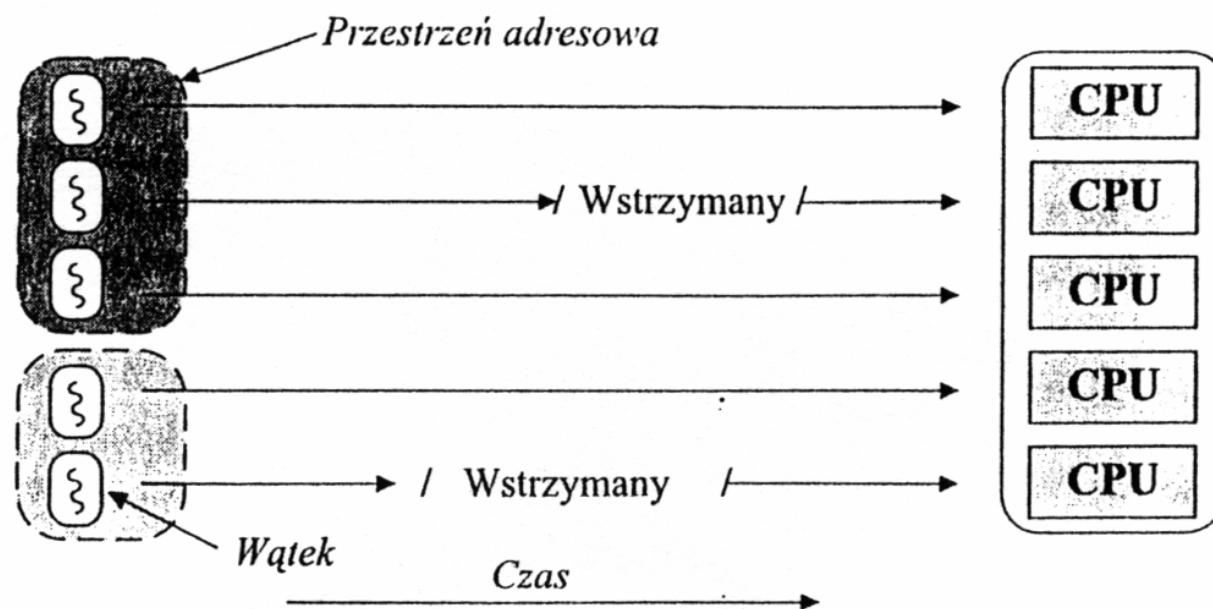
Procesy wielowątkowe w systemie z jednym CPU



Rysunek 3-2. Procesy wielowątkowe w systemie z jednym procesorem

Niektóre architektury CPU posiadają wsparcie sprzętowe wielowątkowości pozwalając na przełączenie między wątkami w skali nanosekund (*Hyper-Threading*)

Procesy wielowątkowe w wieloprocesorze

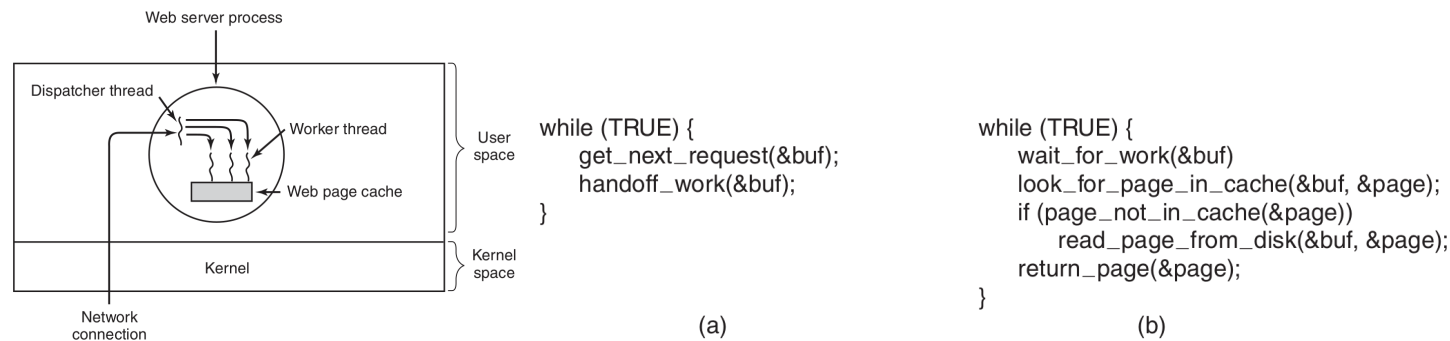


Rysunek 3-3. Procesy wielowątkowe w wieloprocesorze

Nowoczesne procesory mają wiele rdzeni, co pozwala na jednoczesne wykonywanie wielu wątków w sposób rzeczywiście równoległy

Zalety wątków

- przełączanie procesora między wątkami jest łatwiejsze (szybsze) niż między zwykłymi (ciężkimi) procesami
- tworzenie i niszczenie wątków prostsze i szybsze niż procesów (nawet 10-100 razy szybsze)
- lepsze wykorzystanie zasobów systemu komputerowego
- lepsza realizacja przetwarzania współbieżnego na maszynach o pamięci współdzielonej (SMP)

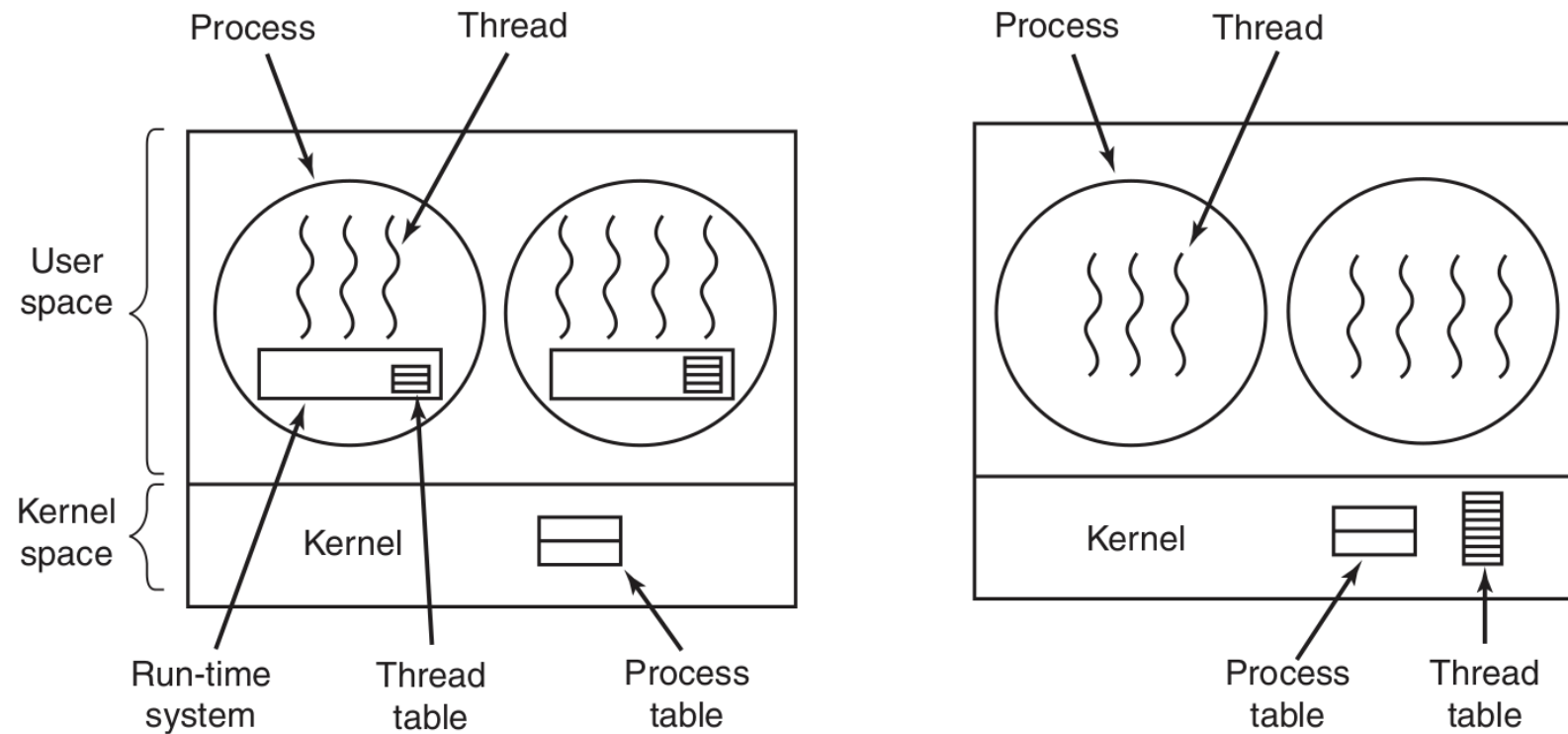


Przykład serwera realizowanego za pomocą wątku dyspozytora i wątków roboczych

Zarządzanie wątkami

- nowo utworzony proces zazwyczaj startuje z pojedynczym wątkiem
- wątek może utworzyć nowy wątek (funkcja `thread_create`) przekazując mu procedurę do wykonania
- wątek po wykonaniu pracy zgłasza zamknięcie (funkcja `thread_exit`)
- wątek może oczekiwać na zakończenie innego wątku (funkcja `thread_join`), jest wówczas w stanie zablokowanym
- wątek może dobrowolnie zrzec się CPU (funkcja `thread_yield`) tak aby dać innym wątkom szansę na wykonanie

Wątki w przestrzeni użytkownika i w przestrzeni jądra



Wątki użytkownika (*user threads*)

- zarządzane przez biblioteki użytkownika, a nie bezpośrednio przez jądro systemu operacyjnego
- mogą być przezroczyste dla jądra, implementacja nie wymaga wsparcia wielowątkowości w systemie
- przykłady: p-wątki (*Pthreads*) wątki wg normy POSIX

Wątki jądra (*kernel threads*)

- zarządzane bezpośrednio przez jądro systemu operacyjnego
- przykłady: *Linux Kernel Threads*, *Windows Threads*

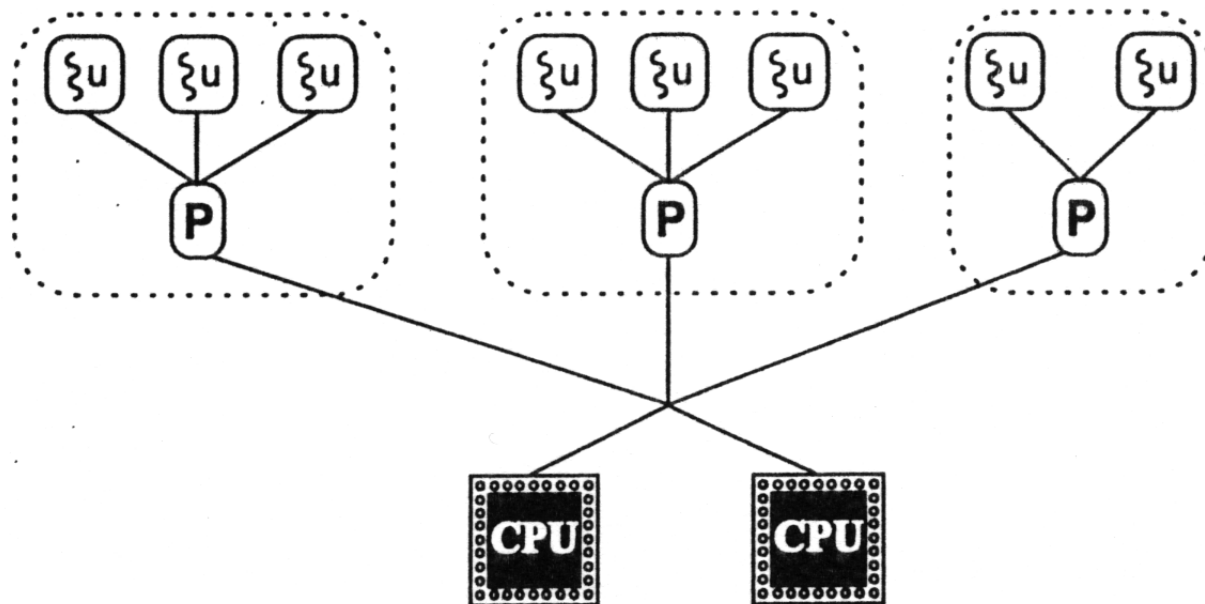
Wątki hybrydowe (*hybrid threads*)

- połączenie wątków użytkownika i jądra, korzystają z mechanizmów wątków użytkownika i jądra, aby zbalansować zalety obu podejść
- procesy lekkie (*lightweight processes LWP*) - działają w przestrzeni użytkownika na pojedynczym wątku jądra, dzielą swoją przestrzeń adresową i zasoby systemowe z innymi LWP w ramach tego samego procesu

Wątki użytkownika

- wątki działające w przestrzeni użytkownika, implementowane przez bibliotekę
- każdy proces przechowuje własną **tablicę wątków**, wskazującą na zasoby każdego wątku (licznik programu, rejestry, stos)
- kontekst wątku zapamiętywany i odtwarzany jest z poziomu użytkownika
- jądro zarządza procesem (nie wie o istnieniu wątków użytkownika), wywołując proces użytkownika wywołuje związane z nim wątki
- wątki użytkownika są wydajne, nie zużywają zasobów jądra (jeśli nie są związane z procesem lekkim)
- przełączanie wątków jest bardzo szybkie w stosunku do wywołań systemowych
- każdy proces może mieć indywidualny schemat zarządzania wątkami

Wątki w przestrzeni użytkownika

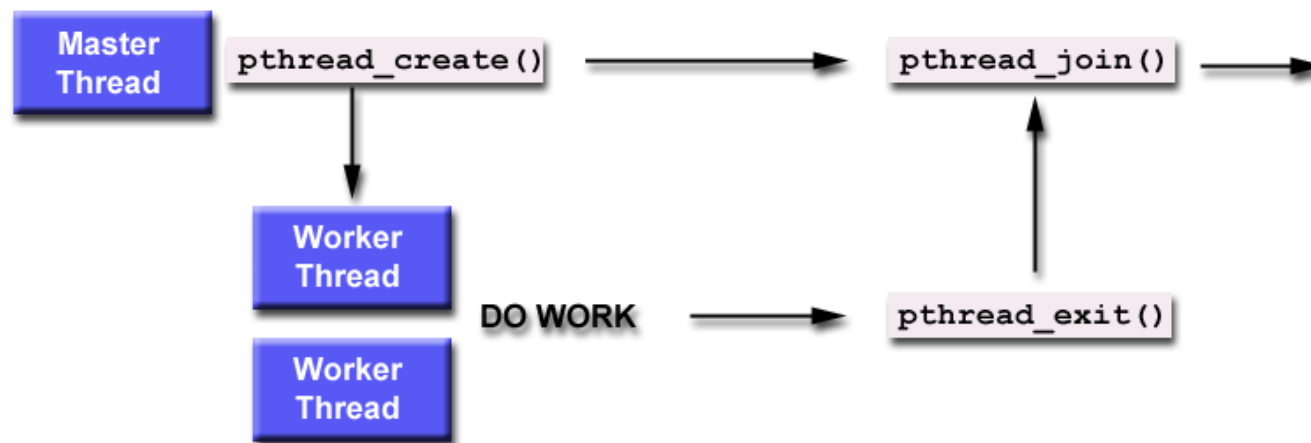


Problemy wątków użytkownika

- biblioteka szereguje wątki, jądro procesy; wątki zwiększają poziom współbieżności, ale nie równoległości
- blokujące operacje (wywołania systemowe, błędy strony) powodują zablokowanie procesu, więc i wszystkich wątków
- wywołanie systemowe generują przerwanie i przejście do trybu jądra, jest to kosztowne
- uruchomiony wątek zajmuje CPU nie dając możliwości działania innym wątkom, dopóki nie zrzeknie się procesora (*yeld*)

p-wątki (pthreads)

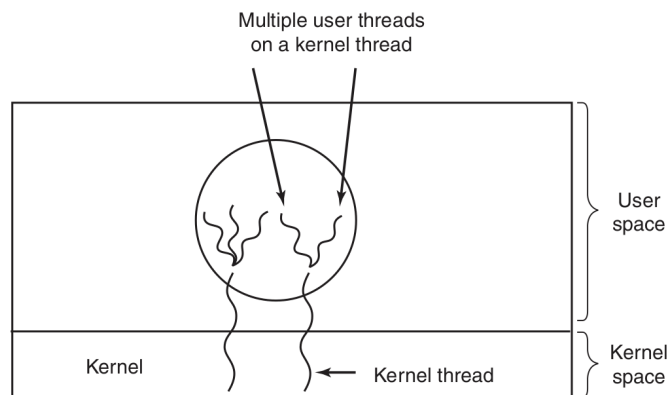
Biblioteka *pthread*s (zgodna ze standardem POSIX) udostępnia abstrakcję wątków całkowicie na poziomie użytkownika, tworzenie, usuwanie, synchronizowanie, szeregowanie oraz zarządzanie wątkami bez udziału jądra



Wątki jądra

- jądro zarządza wątkami, tabela wątków znajduje się w przestrzeni jądra a nie procesu
- tworzenie oraz niszczenie wątków odbywa się przez odwołania do jądra, jest to kosztowniejsze niż w przypadku wątków użytkownika
- wszystkie wywołania blokujące są zaimplementowane jako funkcje systemowe
- w momencie zablokowania wątku system może wznowić wykonywanie innego, gotowego wątku (także z innego procesu)
- wątki jądra nie wymagają implementacji nieblokujących wywołań systemowych

Hybrydowe implementacje wątków

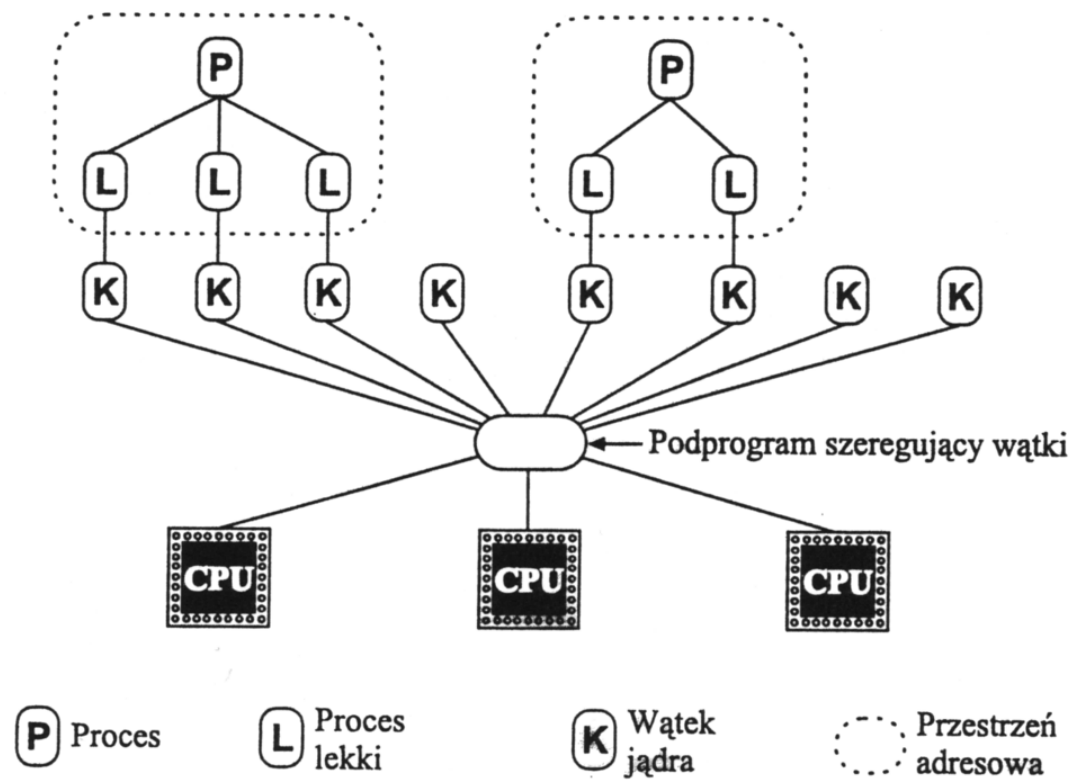


- próba połączenia zalet zarządzania wątkami na poziomie użytkownika i na poziomie jądra
- wątki na poziomie jądra są zwielokrotniane na poziomie użytkownika
- programista może wybrać, ile wątków jądra wykorzystać i na ile wątków użytkownika je zwielokrotnić
- jądro systemu zarządza wyłącznie wątkami jądra
- zarządzanie wątkami użytkownika odbywa w ramach procesu, analogicznie do tradycyjnego modelu wątku użytkownika

Procesy lekkie, LWP

- Proces lekki (*light-weight process*, LWP) jest wspieranym przez jądro wątkiem z przestrzeni użytkownika.
- System udostępniając procesy lekkie musi także udostępniać wątki jądra. W każdym procesie może być kilka procesów lekkich, z których każdy jest wspierany przez osobny wątek jądra.
- Procesy lekkie są niezależnie szeregowane, współdzielą przestrzeń adresową, mogą wywoływać funkcje systemowe, które powodują wstrzymanie w oczekiwaniu na wejście-wyjście lub zasób. Mogą się wykonywać na różnych procesorach.
- Operowanie na procesach lekkich jest kosztowne, gdyż wymaga użycia wywołań systemowych (przełączeń trybu). Trzeba zapewnić synchronizację w dostępie do współdzielonych danych.

Procesy, procesy lekkie i wątki



Procesy i wątki: porównanie

- **Null fork:** czas mierzony od utworzenia poprzez jego szeregowanie, wykonywanie, aż do zakończenia procesu (wątku), który wykonuje pustą procedurę (miara obciążenia spowodowanego przez tworzenie procesu (wątku))
- **Signal-Wait:** czas potrzebny, aby proces (wątek) przesłał sygnał oczekiwania innemu procesowi (wątkowi) i otrzymał odpowiedź zawierającą warunek (miara obciążenia spowodowanego wzajemną synchronizacją)

Czas oczekiwania (μs) na wykonanie operacji przez wątki użytkownika (UT), wątki jądra (KT) i procesy (P).¹⁹

	UT	KT	P
Null fork	34	948	11300
Signal-wait	37	441	1840

¹⁹VAX, system uniksopodobny

Tworzenie procesu w UNIX/Linux

Funkcja systemowa `fork()` tworzy nowy proces

- przydziela nowemu procesowi pozycję w tablicy procesów
- przydziela procesowi potomnemu unikatowy identyfikator PID
- tworzy logiczną kopię procesu macierzystego (ew. zapewniając współdzielenie segmentów instrukcji, itp.); kopiowanie przy zapisie *copy-on-write* odkłada tworzenie rzeczywistej kopii, do momentu gdy fragment pamięci zostanie zmieniony
- zwiększa plikom związanym z tym procesem liczniki w tablicy plików i i-węzłów
- przekazuje identyfikator potomka procesowi macierzystemu i wartość zero procesowi potomnemu

Tworzenie procesu w UNIX/Linux

Wywołanie systemowe `exec()` umieszcza w obszarze pamięci procesu kopię pliku wykonywalnego

- aktualny proces zaczyna wykonywać nowy program
- zawartość kontekstu poziomu użytkownika staje się niedostępna, z wyjątkiem parametrów `exec()`, które jądro kopiuje ze starej do nowej przestrzeni adresowej
- w bibliotece systemowej wywołanie `exec()` realizowane przez funkcję `execve()` lub odmiany tej funkcji

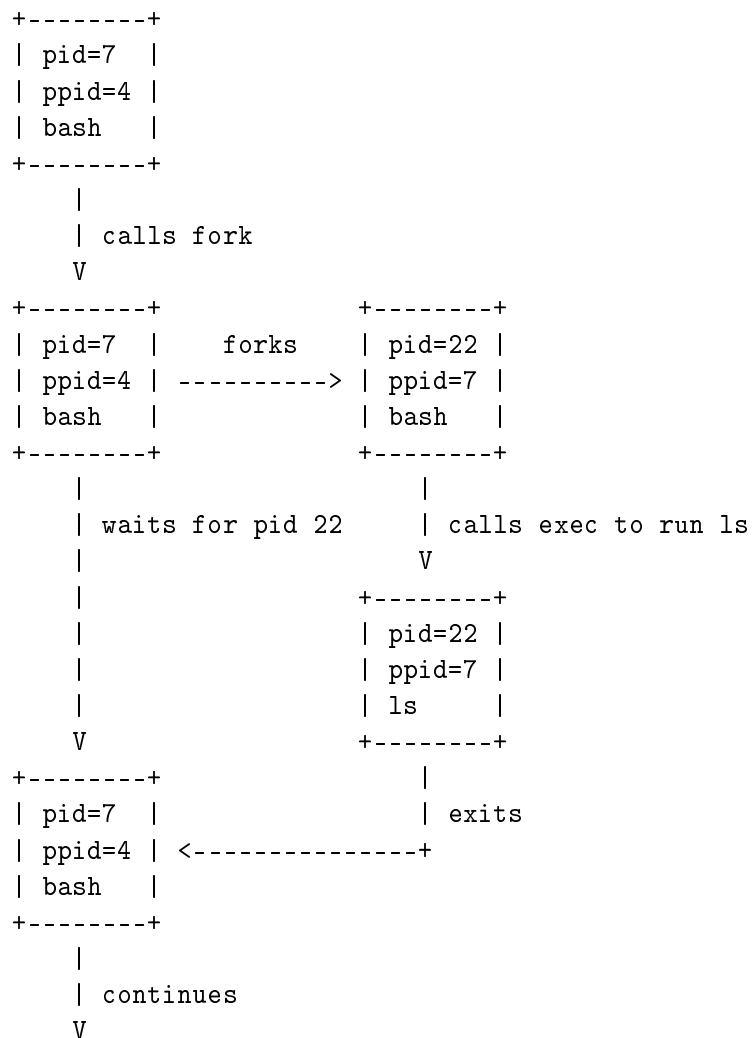
Procesy: tworzenie

Implementacja prostej powłoki

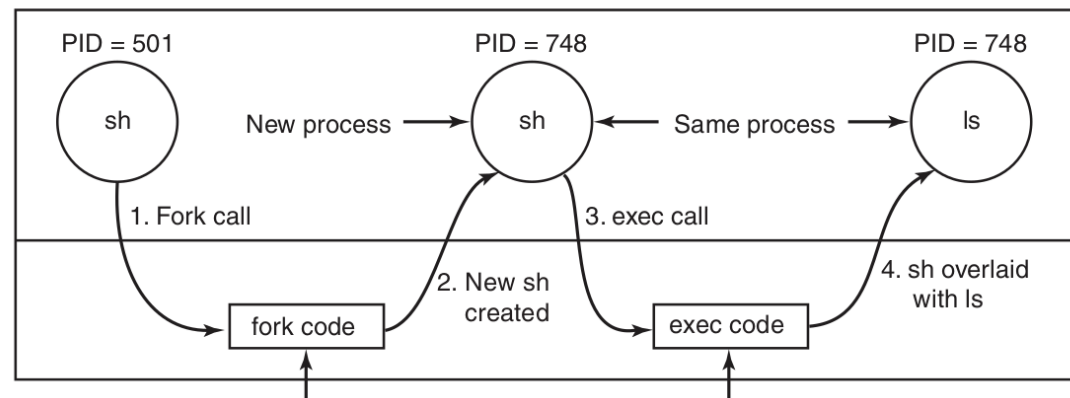
```
while(TRUE) {
    type_command();
    read_command(command, params);

    pid = fork();
    if (pid < 0) {
        printf("Unable to fork()");
        continue;
    }

    if (pid != 0) {
        /* rodzic czeka na potomka */
        waitpid(-1. &status, 0);
    } else {
        /* potomek wykonuje pracę */
        execve(command, params, 0);
    }
}
```



Uruchomienie procesu w powłoce

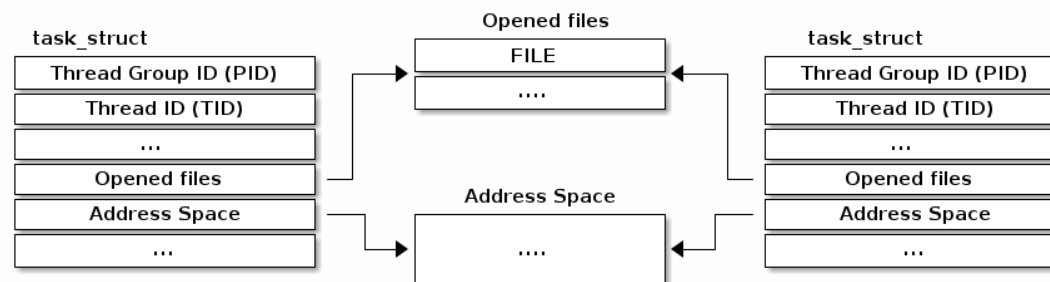


1. przydziela pamięć dla struktury procesu potomnego
2. wypełnia strukturę zadania potomnego danymi rodzica
3. przydziela pamięć dla stosu i przestrzeni użytkownika procesu potomnego
4. wypełnia przestrzeń użytkownika procesu potomnego danymi procesu rodzica
5. przydziela procesowi potomnemu PID
6. ustawia współdzielenie tekstu z rodzicem
7. kopiuje tablice stron danych i stosu
8. ustawia współdzielenie plików z rodzicem
9. kopiuje rejestry rodzica do dziecka
1. odnajduje program wykonywalny
2. sprawdza uprawnienia wykonania
3. odczytuje i weryfikuje nagłówek programu
4. kopiuje argumenty i łańcuchy środowiska do jądra
5. zwalnia starą przestrzeń adresową
6. przydziela nową przestrzeń adresową
7. kopiuje argumenty i łańcuchy środowiska na stos
8. zeruje sygnały
9. inicjuje rejestry

Procesy: zarządzanie w systemie GNU/Linux

W jądrze linuxa procesy (i wątki) reprezentowane są w postaci **zadań** (*task*) przez strukturę danych (*task_struct*)

- struktura zadań reprezentuje kontekst wykonawczy
- zasoby zadania nie są osadzone w strukturze ale struktura posiada wskaźniki do zasobów
- jednowątkowy proces użytkownika to pojedyncze zadanie, wielowątkowy proces to wiele zadań (po jednym na wątek) ale dzielących wspólne zasoby



Atrybuty struktury zadań `task_struct`

- informacje o zadaniu (*task info*): jednoznaczny identyfikator zadania (TID), informacje o związkach rodzic/dziecko pomiędzy zadaniami, identyfikator grupy zadań (TGID), który pozwala identyfikować zadania powiązane procesem
- przestrzeń adresowa (*address space*): definiuje przestrzeń wirtualną zadania
- informacje o przydziale procesora (*scheduling info*): status zadania, politykę przydziału procesora, parametry tej polityki (priorytet dynamiczny), flaga *need_resched*
- informacja o programie wykonywalnym (*executable info*): identyfikuje plik, który jest obrazem wykonywanego zadania
- informacja o sygnałach (*signal info*): zawiera dane związane z sygnałami i ich obsługą (*signal handler table*, *pending signal mask*, *signal queue*)

- dane uwierzytelniające (*credentials*): dane określające prawa i przywileje zadania (UID, GID, maska określająca prawa dostępu do urządzeń)
- informacje księgowe (*accounting info*): czas utworzenia zadania, czas zużyty w trybie jądra i użytkownika, liczba błędów stron, itp.
- ograniczenia zasobów (*resource limits*): parametry określające maksymalną wielkość pliku *core*, czas wykonywania się zadania, wykorzystywaną pamięć, liczbę otwartych plików, itp.
- informacja o systemie plików (*filesystem info*): domyślna maska określająca prawa dostępu przy tworzeniu plików (*umask*), bieżący katalog
- tablica otwartych plików (*open file table*): tablica plików otwartych przez zadanie
- różne (*miscellaneous*): dodatkowe atrybuty zadania potrzebne do jego prawidłowego wykonywania się, np. terminal sterujący

Jądro systemu GNU/Linux: tworzenie zadania

Wywołanie systemowe `clone()` tworzy nowe zadania (wątki lub procesy)

```
pid = clone(function, stack_ptr, sharing_flags, arg);
```

Uruchamia funkcję w kontekście nowego wątku, który współdzieli (lub nie) zasoby zadania zależnie od wartości `sharing_flags`

flagi	współdzielone zasoby	gdy ustawiona	gdy nie ustawiona
CLONE_VM	przestrzeń adresowa	powstaje nowy wątek	powstaje nowy proces
CLONE_FS	informacje o systemie plików katalog roboczy, główny, umask	wspólny katalog roboczy	niezależny katalog roboczy
CLONE_FILES	tablica otwartych plików	współdzielenie deskryptorów	kopie deskryptorów
CLONE_SIGHAND	tablica obsługi sygnałów	współdzieli	kopiuje tablicę
CLONE_PARENT	informacje o rodzicu	nowy wątek współdzieli rodzica	nowy wątek staje się potomkiem
CLONE_PID	PID	wątek ma ten sam PID	proces uzyskuje nowy PID

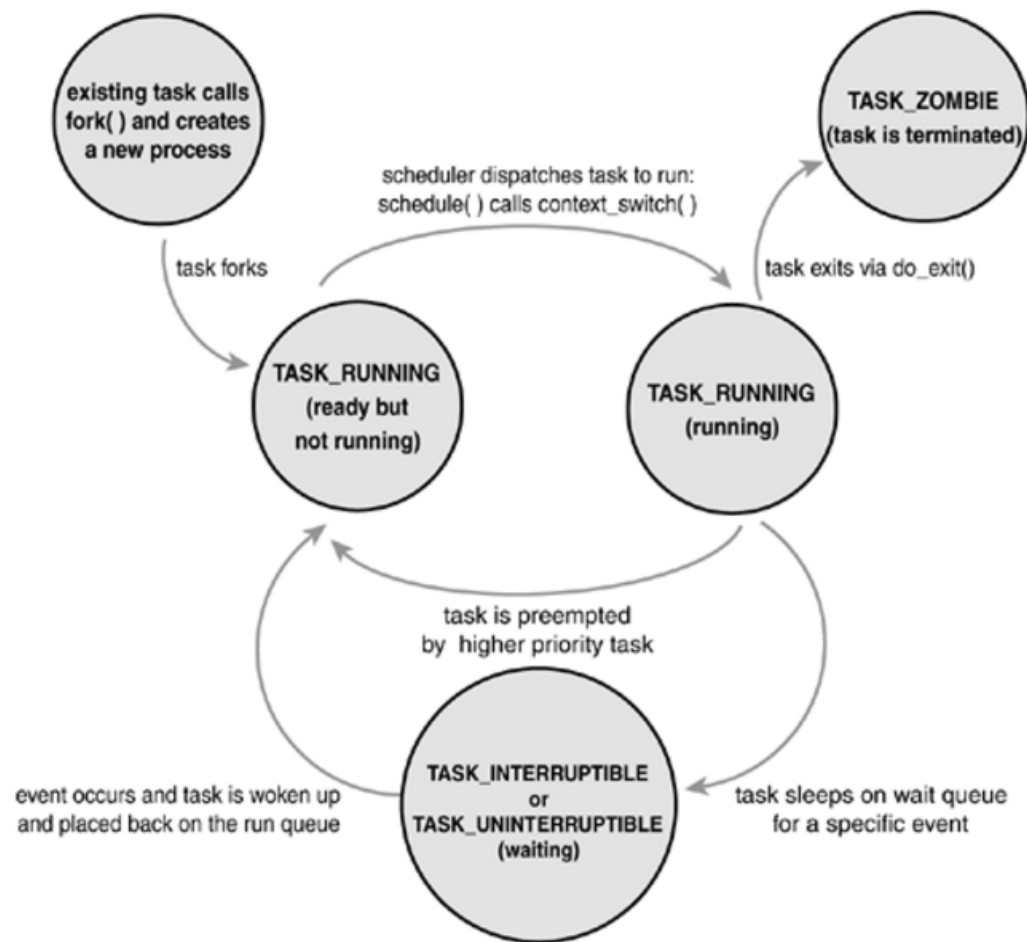
Zarówno wywołanie systemowe `fork()`, jak i funkcja `pthread_create()` korzystają z implementacji `clone()`

Stany procesów wg *Linux kernel*

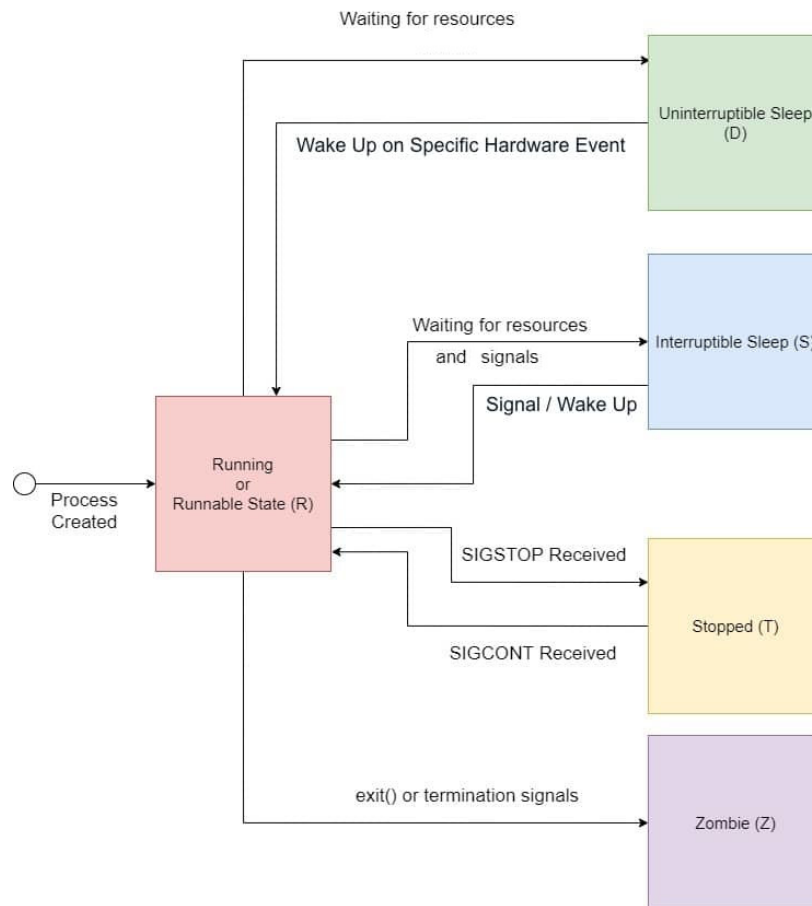
- `TASK_RUNNING` – proces albo się wykonuje, albo czeka na wykonanie w kolejce procesów gotowych.
- `TASK_INTERRUPTIBLE` – proces jest wstrzymany do czasu zajścia określonego zdarzenia. Proces może zostać zbudzony przez przerwania sprzętowe, zwolnienie jakiegoś zasobu systemowego, na który proces czeka, albo dostarczenie sygnału.
- `TASK_UNINTERRUPTIBLE` – wybudzenie może nastąpić po uzyskaniu zasobu, na który proces czekał. Sygnały są ignorowane ²⁰
- `TASK_ZOMBIE` – wykonanie procesu zostało zakończone, ale proces rodzica nie użył jeszcze wywołania systemowego `wait()` lub `waitpid()`, które zwraca informację o przerwany procesie
- `TASK_STOPPED` – wykonanie procesu zostało zatrzymane (wskutek odebrania sygnału `SIGSTOP`, `SIGTSTP`, `SIGTTIN`, `SIGTTOU`).

²⁰`TASK_KILLABLE`, alternatywny stan `TASK_UNINTERRUPTIBLE` reagujący na krytyczne sygnały wprowadzony w łatce do jądra 2.6.25

Stany procesów



Stany procesów



Obserwowanie stanów: polecenie ps

```
$ ps a
  PID TTY          STAT TIME COMMAND
  5270 tty2      SNsl+   0:00 /usr/libexec/gdm-x-session --run-script env GNOME_SHELL_SESSION_MODE=pop /usr/bin/gnome-s
  5272 tty2      S<l+    59:28 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm/Xauthority -nolisten tcp -bac
  5924 pts/0      SNs     0:37 /usr/bin/zsh
1680036 pts/7      TNs+    0:00 zsh -Z -g
1871214 pts/5      TN       0:00 pdflatex so05-procesy.tex
1934828 pts/5      SNLl+   0:36 gimp process-finite-state-machine.jpeg
1940687 pts/0      SN       0:00 /usr/bin/zsh
1940688 pts/0      RN+     0:00 ps a
```

Stany procesów wg man ps

R (running or runnable) – wykonujący się lub zdolny do pracy (oczekujący w kolejce)	X (dead) – (should never be seen)
S (interruptible sleep) – śpiący (blocked)	< (high priority process) – proces o wysokim priorytecie
D (uninterruptable sleep) – nieprzerywalny sen (zwykle I/O)	N (low priority process) – proces o niskim priorytecie
T – zatrzymany przez sygnał sterujący	L (pages locked into memory) – proces ze stronami uwięzionymi w pamięci
t – zatrzymany przez program śledzący (debugger)	s – leader sesji
Z (defunct) “zombie” – zakończony, ale nie zebrany przez proces rodzica	l – proces wielowątkowy
	+ – proces pierwszego planu

Procesy i wątki: obserwowanie procesów wielowątkowych

```
$ ps -eo stat,pid,ppid,tgid,tid,lwp,nlwp,vsz,rsz,cmd
STAT      PID      PPID      TGID      TID      LWP NLWP      VSZ    RSZ  CMD
...
SN1  1217879 1217875 1217879 1217879 1217879      4 321996 29420 /usr/bin/gnome-terminal.real --wait
...
```

```
$ ps -m -o stat,pid,ppid,tgid,tid,lwp,nlwp,vsz,rsz,cmd -p 1217879
STAT      PID      PPID      TGID      TID      LWP NLWP      VSZ    RSZ  CMD
-    1217879 1217875 1217879      -      -      4 321996 29420 /usr/bin/gnome-terminal.real --wait
SN1      -      -      - 1217879 1217879      -      -      - -
SN1      -      -      - 1217880 1217880      -      -      - -
SN1      -      -      - 1217881 1217881      -      -      - -
SN1      -      -      - 1217882 1217882      -      -      - -
```

Z dokumentacji ps:

tgid - identyfikator grupy wątków, do której należy zadanie (alias pid)

tid - unikalny numer reprezentujący jednostkę wykonywalną (*dispatchable entity*) (alias lwp)

lwp - ID lekkiego procesu (wątku) jednostki wykonywalnej (alias tid)

rsz - rozmiar zestawu rezydentnego (*resident set size*), czyli nieswapowana pamięć fizyczna, którą zadanie wykorzystało w kilobajtach (alias rsz)

vsz - rozmiar pamięci wirtualnej procesu w KiB

Jądro nie jest procesem w tradycyjnym sensie, ale zarządcą procesów.

- działają w kontekście procesu, mogą wykonywać operacje blokujące
- działają wyłącznie w trybie jądra (w przestrzeni adresowej jądra)
- nie komunikują się z użytkownikami (nie trzeba terminali)
- tworzone są w chwili startu systemu i działają do czasu wyłączenia systemu

```
$ ps 1 -p 2 --ppid 2
```

F	UID	PID	PPID	PRI	NI	VSZ	RSS	WCHAN	STAT	TTY	TIME	COMMAND
1	0	2	0	20	0	0	0	-	S	?	0:00	[kthreadd]
1	0	3	2	20	0	0	0	-	S	?	0:00	[pool_workqueue_release]
1	0	4	2	0	-20	0	0	-	I<	?	0:00	[kworker/R-rcu_g]
1	0	5	2	0	-20	0	0	-	I<	?	0:00	[kworker/R-slub_]
1	0	6	2	0	-20	0	0	-	I<	?	0:00	[kworker/R-netns]
1	0	8	2	0	-20	0	0	-	I<	?	0:00	[kworker/0:0H-events_highpri]
1	0	11	2	0	-20	0	0	-	I<	?	0:00	[kworker/R-mm_pe]
1	0	12	2	20	0	0	0	-	I	?	0:00	[rcu_tasks_kthread]
1	0	13	2	20	0	0	0	-	I	?	0:00	[rcu_tasks_rude_kthread]

Blokowanie wątku:

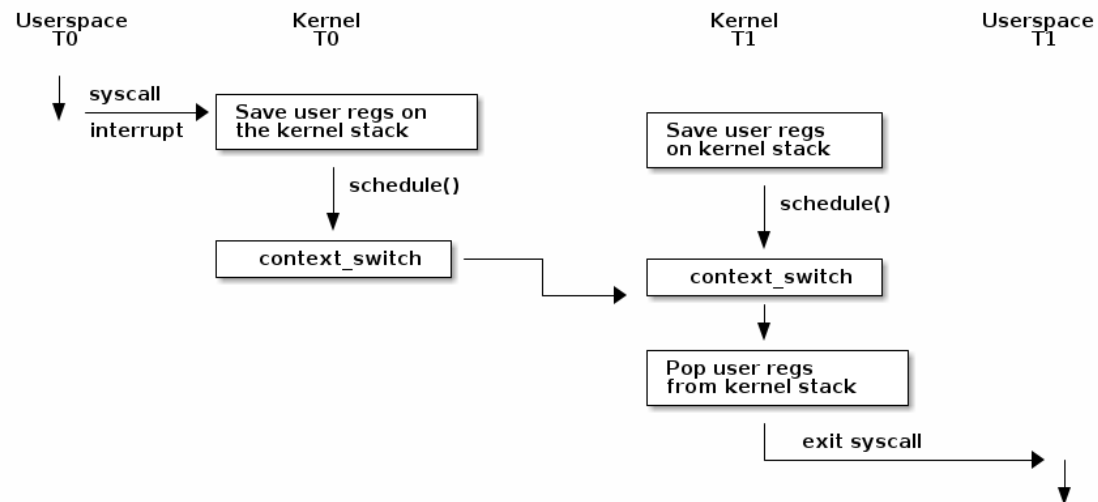
- ustaw bieżący stan wątku na `TASK_UINTERRUPTIBLE` lub `TASK_INTERRUPTIBLE`
- dodaj zadanie do kolejki zadań oczekujących
- wywołaj planistę, który pobierze nowe zadanie z kolejki zadań gotowych
- wykonaj przełączenie kontekstu na nowe zadanie

Budzenie zadania

- wybierz zadanie z kolejki zadań oczekujących
- ustaw stan zadania na `TASK_RUNNING`
- wstaw zadanie do kolejki zadań gotowych
- w systemie SMP, każdy procesor ma własną kolejkę, kolejki muszą być zrównoważone a procesory powiadomione sygnałami

Przełączenie kontekstu

- przed zmianą konte
- przejście do trybu j
- rejestry przestrz
- w tym momencie wątkiem jądra (n
- wygaś przydzielon



Wywłaszczanie procesu

Algorytm szeregujący zwany **planistą** lub **dyspozytorem** może wstrzymać aktualnie wykonywane zadanie (proces lub wątek), aby umożliwić działanie innemu zadaniu.

Do wywłaszczenia może dojść gdy:

- proces wchodzi w stan TASK_RUNNING i jego priorytet jest wyższy od priorytetu procesu właśnie zawłaszczającego procesor
- kiedy proces wyczerpie swój kwant czasu
- wywłaszczenie procesu użytkownika może nastąpić przy powrocie do przestrzeni użytkownika z wywołania systemowego lub z procedury obsługi przerwania
- kiedy jednostka centralna wykonuje kod w trybie użytkownika

Wywłaszczenie jądra

Czy proces może być wywłaszczony, jeśli przebywa w trybie jądra?

- jądra uniksowe są **wielobieżne** (wielowejściowe, *reentrant*): kilka procesów może się wykonywać w trybie jądra w tym samym czasie
- w systemie jednoprocessorowym tylko jeden proces może działać, inne mogą czekać na CPU lub na zakończenie operacji wej/wyj (będąc zablokowanymi w trybie jądra)
- wystąpienie przerwania sprzętowego pozwala jądru wielobieżnemu na zatrzymanie procesu, nawet jeśli znajduje się on w trybie jądra. Wpływa to na zwiększenie szybkości obsługi urządzeń zewnętrznych.
- Linux (z jądrem w wersji ≤ 2.4) jest systemem operacyjnym z wywłaszczaniem procesów, ale bez wywłaszczania jądra. Jądro ≥ 2.6 jest już wielowejściowe, umożliwia wywłaszczenie zadania działającego w trybie jądra i wykonującego wywołanie systemowe

Algorytmy przydziału procesora

Planista (ang. scheduler) lub dyspozytor – część systemu operacyjnego odpowiedzialna za przydzielanie czasu procesora w ramach przełączania zadań. Jeśli dwa lub więcej procesów znajdują się w stanie *gotowy do wykonania*, to o kolejności przydziału CPU decyduje **algorytm szeregowania**

Planista powinien gwarantować:

- sprawiedliwość przydziału CPU
- dobrą wydajność CPU
- mały czas odpowiedzi
- mały czas przetwarzania (*turnaround time*)
- dużą przepustowość (*throughput*)

Procesy ograniczone obliczeniowo i operacjami wej-wyj

Klasyfikacja procesów I:

- zorientowane na wejście-wyjście (*I/O-bound*)
Wymagają szybkiego przydziału czasu procesora i częstszych przełączeń kontekstu, aby efektywnie obsługiwać operacje wejścia-wyjścia.
- zorientowane na obliczenia (*CPU-bound*)
Wymagają dłuższego czasu procesora i rzadszych przełączeń kontekstu, aby zmaksymalizować wydajność obliczeniową

Wraz ze wzrostem szybkości CPU procesy mają tendencję stawać się bardziej ograniczone wej-wyj

W systemach Unix/Linux planista zwykle jawnie faworyzuje procesy ograniczone przez operacje wejścia-wyjścia.

Szeregowanie bez wywłaszczenia

Algorytm szeregowania **bez wywłaszczania** (*non-preemptive*) wybiera proces do uruchomienia a następnie pozwala mu działać do czasu zablokowania (np. operacja we-wyj) lub do momentu, gdy proces dobrowolnie zwolni CPU

- prosty w implementacji, małe narzuty związane z przełączaniem kontekstu
- przerwanie zegarowe nie powoduje podjęcia decyzji o szeregowaniu
- wykorzystywany w systemach wsadowych, nie wymagających interakcji z użytkownikiem
- może prowadzić do mniejszej responsywności systemu dla procesów interaktywnych
- ryzyko głodzenia procesów o niższym priorytecie

Szeregowanie z wywłaszczaniem

Algorytm szeregowania **z wywłaszczaniem** (*preemptive*) wybiera proces i pozwala mu działać maksymalnie przez ustalony czas

- procesy mogą być przerwane (wywłaszczane) w dowolnym momencie, zazwyczaj przez przerwania zegara (*timer interrupts*), aby dać miejsce procesom o wyższym priorytecie
- lepsza responsywność dla procesów interaktywnych
- sprawiedliwsze przydzielanie czasu procesora w systemach wielozadaniowych
- trudniejsze w implementacji
- możliwe większe narzuty związane z częstymi przełączaniami kontekstu

Cele algorytmów szeregowania w różnych środowiskach

Wszystkie systemy

- sprawiedliwość - przydział procesom uczciwego dostępu do CPU
- równowaga - wszystkie części systemu powinny być zajęte
- wymuszenie polityki szeregowania

Systemy interaktywne

- czas odpowiedzi - responsywność
- proporcjonalność - spełnienie oczekiwań użytkowników dotyczących czasu wykonania

Systemy wsadowe

- przepustowość - maksymalizacja liczby wykonanych zadań na godzinę
- czas cyklu przetwarzania - minimalizacja czasu pomiędzy rozpoczęciem i zakończeniem procesu
- wykorzystanie CPU - maksymalizacja zajętości

Systemy czasu rzeczywistego

- dotrzymywanie terminów - unikanie utraty danych
- przewidywalność - unikanie degradacji jakości w systemach multimedialnych

Planowanie przydziału procesora

- pierwszy nadszedł, pierwszy obsłużony (FCFS, *First-Come, First-Served*)
- najkrótsze zadanie najpierw (SJF, *shortest job first*)
- rotacyjne, cykliczne (*Round-Robin*)
- priorytetowe (polityka planowania i mechanizm planowania)
- dwupoziomowe, wielopoziomowe

Pierwszy nadszedł, pierwszy obsłużony (FCFS)

First-Come, First-Served

- pojedyncza kolejka gotowych procesów
- procesy są wykonywane w kolejności, w jakiej zgłaszają się do systemu, bez względu na ich czas trwania czy priorytet
- proces jest wywłaszczany w momencie zablokowania, kolejny proces zostaje wybrany
- przy wznowieniu pracy zablokowanego procesu trafia na koniec kolejki
- prosty w implementacji i sprawiedliwy, bo każdy proces jest obsługiwany w kolejności zgłoszeń
- długie procesy mogą nadmiernie blokować CPU, na niekorzyść krótkich zadań
- przydatny w systemach wsadowych, nie jest optymalny dla systemów interaktywnych, gdzie responsywność jest kluczowa

Najkrótsze Zadanie Najpierw (SJF, *Shortest Job First*)

- procesy są sortowane i wykonywane w kolejności ich przewidywanego czasu wykonania, od najkrótszego do najdłuższego
- proces o najkrótszym przewidywanym czasie wykonania jest wybierany jako pierwszy do uruchomienia
- minimalizuje średni czas oczekiwania, ponieważ krótsze zadania są obsługiwane szybciej, co zmniejsza opóźnienia w kolejce
- w praktyce trudno jest dokładnie oszacować czas wykonania każdego zadania
- dłuższe procesy mogą być stale opóźniane (głodzenie), jeśli nowe, krótsze zadania ciągle się pojawiają
- efektywne w systemach, gdzie czas wykonania procesów jest znany lub może być oszacowany z dużą dokładnością (systemy wsadowe)

Najkrótsze Zadanie Najpierw (SJF, *Shortest Job First*)



Figure 2-41. An example of shortest-job-first scheduling. (a) Running four jobs in the original order. (b) Running them in shortest job first order.

Czas oczekiwania w oryginalnym porządku:

8, 12, 16, 20, średnio 14

Czas oczekiwania w kolejności od najkrótszego zadania:

4, 8, 12, 20, średnio 11

Algorytm **najkrótszego pozostałego czasu** (SRT, *Shortest Remaining Time*) - algorytm SJF z wywłaszczeniem, które następuje w momencie pojawienia się nowego zadania o krótszym spodziewanym czasie niż pozostały czas bieżącego zadania

Algorytm **najkrótszego następnego procesu** (SPN, *Shortest Process Next*) - używa historii uruchomień i zachowania procesów do estymacji czasu wykonywania. Stosowany w systemach interaktywnych

Np. średnia ważona aktualnego czasu i poprzedniego (*aging*)

$$T_{n+1} = aT_{n-1} + a(1 - T_n)$$

Algorytm rotacyjny (*Round-Robin*)

- procesy gotowe do wykonania są zorganizowane w postaci listy, każdy proces otrzymuje kwant czasu
- gdy kwant dobiegnie końca proces jest wyłuszczany i umieszczany na końcu listy
- proces jest również wyłuszczany w momencie zablokowania
- kwant czasu ok. 20-50 msec, gdy za krótki może powodować spadek wydajności z powodu zbyt częstych przełączeń kontekstu, zbyt długi kwant powoduje słabą responsywność krótkich interaktywnych zadań

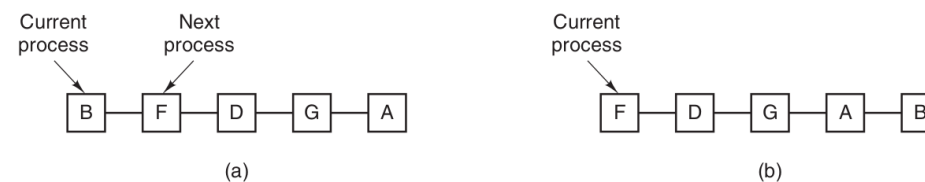


Figure 2-42. Round-robin scheduling. (a) The list of runnable processes. (b) The list of runnable processes after *B* uses up its quantum.

Planowanie priorytetowe

- decyzja planisty zależna od **priorytetów** przypisanych do zadań, ważniejsze zadania otrzymują wyższe priorytety i mają pierwszeństwo
- priorytety mogą być przydzielane **statycznie** lub **dynamicznie**
- statyczne priorytety są przypisywane na podstawie właściwości procesu i nie zmieniają się w trakcie jego życia
- dynamiczne priorytety mogą zmieniać się w zależności od zachowania procesu, np. procesy interaktywne (które często czekają na dane od użytkownika) mogą być promowane przez podwyższenie priorytetu aby zwiększyć responsywność
- procesy mogą zostać pogrupowane względem priorytetów tworząc **klasy priorytetów**. Planowanie między klasami odbywa się zgodnie z priorytetami, zaś w ramach konkretnej klasy wg. wybranego algorytmu szeregowania (np. cyklicznego)

Klasy priorytetów

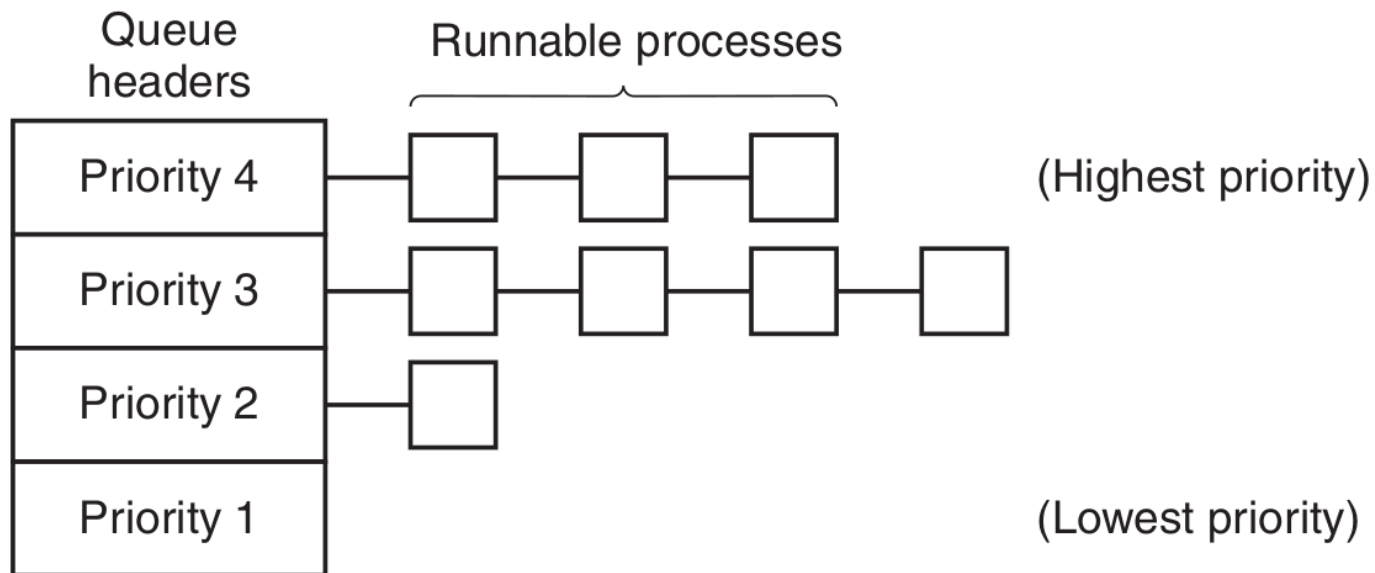


Figure 2-43. A scheduling algorithm with four priority classes.

Algorytm szeregowania w Linuksie

- szeregowaniu podlegają wątki jądra
- 140 priorytetów: $[0,139]$, gdzie niższa wartość oznacza wyższy priorytet
- klasa zadań czasu rzeczywistego: priorytety statyczne $[0,99]$
polityki planowania: `SCHED_FIFO`, `SCHED_RR`
- zadania normalne: priorytety dynamiczne $[100,139]$
polityki planowania: `SCHED_OTHER`
- **planista $O(1)$** używany od wersji jądra 2.5 (11-2001)
- **planista całkowicie sprawiedliwy** (`CFS`, *Completely Fair Scheduler*) o złożoności $O(\log N)$ pojawił się w wersji 2.6.23 (10-2007)

,

Priorytety czasu rzeczywistego

- priorytety zadań czasu rzeczywistego są z zakresu 0-99, priorytety nie ulegają zmianie w czasie wykonywania
- zadania wykonywane są ściśle w kolejności malejących priorytetów, najpierw zadania czasu rzeczywistego, potem pozostałe zadania

Klasy priorytetów czasu rzeczywistego:

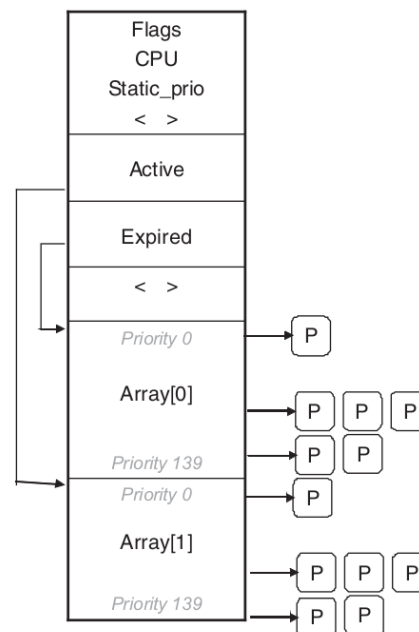
- SCHED_FIFO (*First-In, First-Out*) – szeregowanie bez wywłaszczenia, zadania są wykonywane w kolejności ich przybycia, aż do zakończenia lub do momentu pojawienia się gotowego zadania o wyższym priorytecie
- SCHED_RR (*Round-Robin*) - szeregowanie rotacyjne, każdy proces dostaje równy kwant czasu do wykorzystania, po wyczerpaniu kwantu czasu proces jest przesuwany na koniec kolejki i ponownie czeka na swoją kolej

Priorytety dynamiczne w planiście O(1)

- klasa SCHED_OTHER - standardowe procesy użytkownika o priorytetach dynamicznych z zakresu 100-139
- początkowa wartość priorytetu dynamicznego jest określana przez wartość *poziomu uprzejmości*
- poziomy uprzejmości (*nice*) od -20 do +19 są mapowane na wartości od 100 do 139, domyślna wartość uprzejmości 0 to priorytet 120
- wartość priorytetu procesu jest zmieniana w granicach ± 5 w zależności od interaktywności zadania (stosunek czasu jaki zadanie spędza w zawieszeniu do czasu aktywności jest miarą uzależnienia zadania od operacji wejścia-wyjścia)
- wielkość kwantu czasu procesu zależy od priorytetu dynamicznego, procesy o wyższym priorytecie otrzymują dłuższy kwant czasu
- domyślnie proces otrzymuje kwant 100 ms. Ten kwant może być zwiększony do 800 ms (100) lub zmniejszony do 5 ms (139).
Nowy proces dziedziczy kwant po swoim rodzicu.

Planista $O(1)$

- struktura zarządzania aktywnymi zadaniami (*runqueue*) zawiera dwie tablice o rozmiarze 140: zadania aktywne (*active*) i wygaste (*expired*)
- każdy element tablicy wskazuje na początek kolejki zadań o wspólnym priorytecie



(a) Per CPU runqueue in the Linux $O(1)$ scheduler

- planista wybiera pierwsze zadanie z listy aktywnych o najwyższym priorytecie
- po wyczerpaniu kwantu czasu zadanie jest umieszczane na liście wygasłych (możliwa jest zmiana priorytetu)
- jeśli zadanie zostanie zablokowane przed upływem kwantu czasu to powraca do listy aktywnych z odpowiednio skróconym kwantem
- gdy wszystkie listy aktywnych zadań się wyczerpią, następuje zamiana tablic, zadania wygasłe stają się znów aktywne
- wybór zadania odbywa się w stałym czasie, niezależnie od liczby zadań
- algorytm gwarantuje, że wszystkie zadania, nawet o niskim priorytecie, mają szansę wykonania
- główną wadą algorytmu jest skomplikowana heurystyka ustalania priorytetów dynamicznych, promująca zadania interaktywne, kosztem zadań obliczeniowych

Co oznacza PRI?

proces/wątek	jądro	"top/htop"	"ps -eo pri"	"ps -eo rtprio"	"ps -el"	"nice"
migration	99	RT	139	99	-40	-
kworker	100	0	39	-	60	-20
bash	120	20	19	-	80	0
khugepaged	139	39	0	-	99	19

	priorytet wg jądra ≥ 100	priorytet wg jądra < 100
top/htop	PRI='priorytet wg jądra' - 100	PRI=RT
ps -eo pri	PRI=139 - 'priorytet wg jądra'	PRI='priorytet wg jądra' + 40
ps -el	PRI='priorytet wg jądra' - 40	PRI='priorytet wg jądra' - 139

Obecne klasy szeregowania w jądrze Linux

Klasy zadań objętych wspólną polityką przydziału CPU:

- czasu rzeczywistego (*real time*): SCHED_RR, SCHED_FIFO
- sprawiedliwa (*fair*): SCHED_NORMAL (dawniej SCHED_OTHER) większość procesów użytkownika, zarządzana przez planistę CFS
- SCHED_BATCH dla zadań obciążających CPU nieinteraktywnych o niskim priorytecie
- bezczynna: SCHED_IDLE dla procesów o bardzo niskim priorytecie (niżej niż +19), które mogą być wykonywane tylko wtedy, gdy system jest bezczynny
- terminowa (*deadline*): SCHED_DEADLINE dla zadań o ścisłych wymaganiach czasowych (systemy czasu rzeczywistego)

Przykład

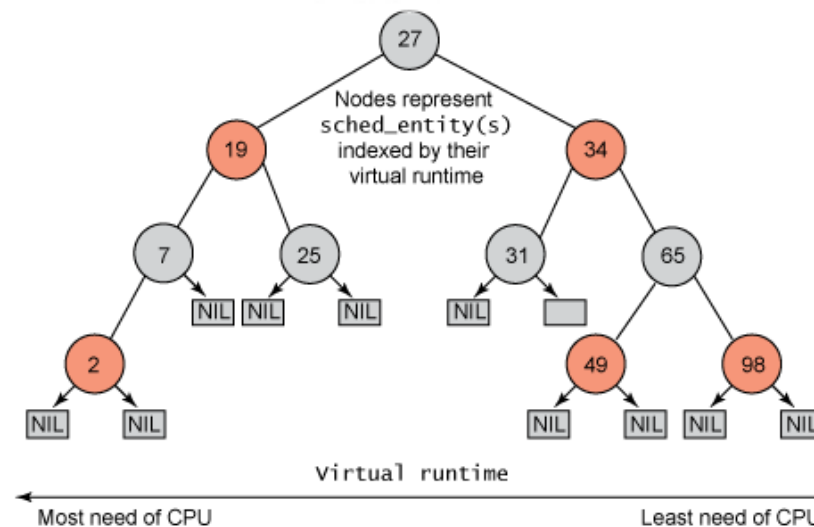
```
$ ps -e -o s,pid,policy,pri,ni,rtprio,command
S      PID POL PRI  NI RTPRIO COMMAND
S        1 TS   19   0      - /sbin/init splash
S        2 TS   19   0      - [kthreadd]
I        4 TS   39 -20      - [kworker/R-rcu_g]
...
S       19 FF  139   -     99 [migration/0]
S     4565 B    4  15      0 /usr/libexec/packagekitd
S     5670 TS    7  12      - /usr/bin/python3 /usr/lib/pop-transition/service.py
R    13056 IDL   0   -      0 /usr/libexec/tracker-miner-fs-3
S    30080 TS   13   6      - /usr/bin/python3 /usr/bin/gnome-terminal --wait
R   464202 TS   19   0      - ps -e -o s,pid,policy,pri,ni,rtprio,command
```

Polecenie `chrt` ustawia lub odczytuje atrybuty szeregowania

```
$ chrt -p 19
pid 19's current scheduling policy: SCHED_FIFO
pid 19's current scheduling priority: 99
```

Planista całkowicie sprawiedliwy CFS

- struktura zarządzania zadaniami (runqueue) jest zorganizowana w postaci drzewa czerwono-czarnego
- zadania w drzewie są uporządkowane w kolejności względem ilości czasu wirtualnego przez jaki zajmowały CPU (wartość `vruntime`)



Planista CFS

- planista zawsze wybiera do wykonania zadanie o najmniejszym czasie wirtualnym (skrajny węzeł z lewej)
- okresowo planista aktualizuje wirtualny czas działającego zadania porównując z wartością czasu kolejnego zadania znajdującego się w drzewie
- jeżeli w drzewie znajduje się węzeł o niższym czasie to następuje wywłaszczenie i wykonywane zadanie zostaje umieszczone ponownie w drzewie we właściwym miejscu
- różnice w priorytetach zadań oraz uprzejmości (nice) są odzwierciedlone w postaci szybkości upływu wirtualnego czasu
- dla zadań o niskim priorytecie czas płynie szybciej, więc szybciej ulegają wywłaszczeniu w stosunku do zadań o wyższym priorytecie

Komunikacja międzyprocesowa (IPC)

Inter-Process Communication

- pamięć współdzielona (*Shared Memory*)
- potoki (*pipes*) - jednokierunkowe przesyłanie strumieni danych
 - potoki nazwane (*named pipes*) - dla procesów niepowiązanych
- sygnały
- gniazda (*sockets*)
- semafony
- pliki
- kolejki komunikatów (*Message Queues*) - przesyłanie asynchroniczne komunikatów za pomocą buforowanych kolejek
- zdalne wywołanie procedur (*remote procedure call*, RPC)

Problemy IPC

- wyścigi (*race conditions*) - wynik działania zależy od nieprzewidywalnego przeplatania się operacji wykonywanych przez różne procesy
- zakleszczenie (*deadlock*) - dwa lub więcej procesów czekaj na zasoby, które są trzymane przez siebie nawzajem, co powoduje, że żaden z procesów nie może kontynuować
- zablokowanie (*livelock*) procesy ciągle zmieniają swoje stany w odpowiedzi na działania innych procesów, nie wykonując jednak żadnej efektywnej pracy
- głodzenie (*starvation*) - proces nie otrzymuje zasobów, których potrzebuje do kontynuowania pracy, z powodu nieustannego przydzielania tych zasobów innym procesom
- problem odwróconych priorytetów (*priority inversion*) - proces o wyższym priorytecie jest blokowany przez proces o niższym priorytecie, który trzyma zasób

Komunikacja międzyprocesowa: sygnały

Sygnały są krótkimi wiadomościami, które można wysyłać do procesu, wątku lub grupy procesów. Z każdym sygnałem jest związana jego nazwa i numer (zależne od platformy)

- są prostą (ograniczoną) formą komunikacji międzyprocesowej zdefiniowaną w normie POSIX (dostępne w Unix, Linux, macOS X)
- są programowymi odpowiednikami przerwania sprzętowych, obsługa sygnału odbywa się w ramach procesu w trybie użytkownika
- niektóre wyjątki sprzętowe powodują dostarczenie sygnału do procesu (SIGBUS, SIGEMT, SIGFPE, SIGILL, SIGSEGV, SIGTRAP)
- informują proces lub wątek o wystąpieniu określonego zdarzenia
- zmuszają proces do przerywania aktualnej pracy i wykonania procedury obsługującej sygnał
- jeżeli proces nie zarejestrował funkcji obsługi sygnału to uruchamiana jest domyślna funkcja obsługi (zazwyczaj zamknięcie procesu)

Sygnały

Jądro używa sygnałów, żeby powiadomić proces (zadanie) o wystąpieniu błędów lub asynchronicznych zdarzeń.

Przykłady:

- naruszenie ochrony pamięci (SIGSEGV, *segmentation violation signal*)
- przerwanie klawiaturowe Ctrl-C (SIGINT), Ctrl-Z (SIGSTOP)
- polecenie `kill`
- funkcje systemowe `kill()`, `pthread_kill()`

Obsługa sygnału

Działania podejmowane przez proces po otrzymaniu sygnału:

1. wyłapanie sygnału i uruchomienie procedury obsługi
2. zignorowanie sygnału
3. działanie domyślne, gdy nie zdefiniowano obsługi
 - przerwanie (Term) - proces jest niszczone
 - zrzut (Core) - tworzony jest plik core²¹ i proces jest niszczone
 - ignorowanie (Ign) - sygnał jest pomijany
 - zatrzymanie (Stop) - proces jest zatrzymywany (przechodzi w stan TASK_STOPPED)
 - kontynuacja (Cont) - proces ze stanu TASK_STOPPED przechodzi w stan TASK_RUNNING

²¹Także core.\$\$ lub /var/lib/systemd/coredump/core.sar.1000...

Standardowe sygnały

Numer	Nazwa	Domyślnie	Opis
1	SIGHUP	Term	zamknięcie terminala kontrolującego
2	SIGINT	Term	przerwanie z klawiatury
3	SIGQUIT	Core	zamknięcie z klawiatury
4	SIGILL	Core	nieprawidłowa instrukcja
5	SIGTRAP	Core	pułapka dla programu śledzącego
6	SIGABRT, SIGIOT	Core	nieprawidłowe zakończenie
7	SIGBUS	Core	błąd szyny (np. niepoprawny adres fizyczny)
8	SIGFPE	Core	wyjątek operacji zmiennoprzecinkowej
9	SIGKILL	Term	wymuszone zakończenie procesu
10	SIGUSR1	Term	do wykorzystania przez proces
11	SIGSEGV	Core	nieprawidłowe odwołanie do pamięci
12	SIGUSR2	Term	do wykorzystania przez proces
13	SIGPIPE	Term	zapis do potoku, którego nikt nie czyta
14	SIGALRM	Term	alarm upłynięcia czasu z funkcji alarm()
15	SIGTERM	Term	zakończenie procesu

Numer	Nazwa	Domyślnie	Opis
16	SIGSTKFLT	Term	błąd stosu
17	SIGCHLD	Ign	proces potomny zakończony lub zatrzymany
18	SIGCONT	Cont	wznawianie zatrzymanego procesu
19	SIGSTOP	Stop	zatrzymanie wykonywania procesu
20	SIGTSTP	Stop	zatrzymanie procesu z terminala (Ctrl+Z)
21	SIGTTIN	Stop	program w tle żąda wejścia z terminala
22	SIGTTOU	Stop	program w tle żąda wyjścia na terminal
23	SIGURG	Ign	pilne dane w gnieździe
24	SIGXCPU	Core	przekroczenie limitu czasu procesora
25	SIGXFSZ	Core	przekroczenie limitu wielkości pliku
26	SIGVTALRM	Term	alarm zegara wirtualnego
27	SIGPROF	Term	alarm zegara programu profilującego
28	SIGWINCH	Ign	zmiana rozmiaru okna (terminala)
29	SIGIO, SIGPOL	Term	gotowość do operacji I/O
30	SIGPWR, SIGLOST	Term	awaria źródła zasilania
31	SIGUNUSED, SIGSYS	Core	nie używane (dla wstecznej kompatybilności)

Sygnały SIGKILL i SIGSTOP nie mogą zostać przechwycone, zablokowane ani ignorowane

Ryzyka sygnałów

- sygnały są asynchroniczne, obsługa sygnałów może powodować wystąpienie wyścigu (*race conditions*)
- inny sygnał może zostać dostarczony do procesu podczas wykonywania procedury obsługi sygnału
- sygnały mogą spowodować przerwanie trwającego (np. zablokowanego) wywołania systemowego
- procedury obsługi sygnałów powinny być napisane w sposób nie powodujący niepożądanych skutków ubocznych (np. zmiana globalnych zmiennych procesu, użycie funkcji niebezpiecznych z punktu widzenia asynchronicznego wykonania)
- z punktu widzenia jądra wykonanie kodu obsługi sygnału jest dokładnie takie samo, jak wykonanie dowolnego innego kodu przestrzeni użytkownika (odpowiedzialność programisty)

Maskowanie sygnałów

- proces ma możliwość zamaskować (zablokować) czasowo wybrane sygnały w celu wykonania krytycznych operacji (maska jest zasobem procesu)
- sygnały standardowe nie są kolejkowane, tylko jeden sygnał danego typu dociera po odblokowaniu
- przy odblokowaniu kilku sygnałów różnego typu kolejność obsługi jest nieokreślona
- proces może zablokować się w oczekiwaniu na sygnał (obsługa synchroniczna)

Sygnały czasu rzeczywistego w Linux

Linux używa 31 standardowych sygnałów i do 32 sygnałów czasu rzeczywistego (od 32 do 64)

Sygnały czasu rzeczywistego:

- nie mają zdefiniowanego zastosowania, programiści mogą ich używać według własnego uznania
- domyślne działanie to przerwanie wykonywania procesu
- mogą być powiązane z danymi dostarczonymi do procesu (liczba lub wskaźnik)
- umożliwiają odczytanie identyfikatora procesu i użytkownika nadawcy
- są kolejowanie a kolejność dostarczenia jest zagwarantowana, im niższy numer sygnału tym wyższy priorytet
- minimalny i maksymalny numer sygnału zdefiniowane przez makro SIGRTMIN i SIGRTMAX (np. SIGRTMIN+1, SIGRTMAX-2)

Wysyłanie sygnałów z poziomu powłoki

- wysyłanie sygnałów do procesów
`kill -SIGNAL PID`
- zatrzymywanie i wznowianie procesów
`kill -STOP PID`
`kill -CONT PID`
- zamknięcie procesu (domyślnie sygnał SIGTERM)
`kill PID`
- zabicie procesu
`pkill -9 top`
- zabicie wszystkiego co się da
`kill -1 -9`

Sygnał SIGKILL/9 nie trafia do wskazanego procesu, ale do procesu `init/systemd`

Zasoby niepodzielne: zasoby systemowe, które nie mogą być współdzielone ani używane równocześnie przez więcej niż jeden proces w danym momencie.

- większość urządzeń zewnętrznych, pliki zapisywalne, obszary danych, które ulegają zmianom

Zasoby podzielne:

- jednostki centralne, pliki tylko do czytania, obszary pamięci, gdzie znajdują się (dzielone) biblioteki, kody programów, itp.

W systemie wieloprocesowym lub wieloprocessorowym (wielowątkowym) mamy do czynienia z (pseudo)równoczesnymi wątkami wykonania.

- Jak zapewnić prawidłowy dostęp procesów do zasobów niepodzielnych?
- Jak unikać zakleszczeń przy dostępie procesów do dwóch lub więcej zasobów równocześnie?

Wyścigi i sekcje krytyczne

Przykład 1: system rezerwacji biletów

Biuro A widzi, że jest wolne
miejsce X i powiadamia o
tym swego klienta

:

Biuro A rezerwuje miejsce X

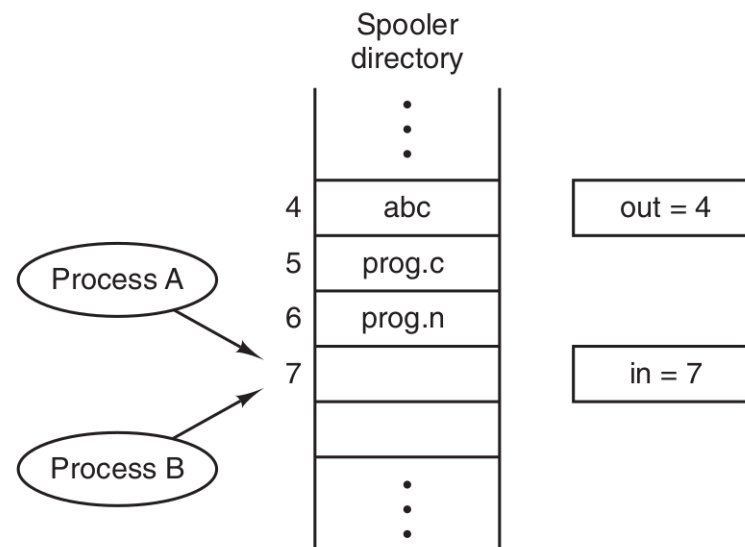
Biuro B widzi, że jest wolne
miejsce X i powiadamia o
tym swego klienta

:

Biuro B rezerwuje miejsce X

Wyścigi i sekcje krytyczne

Przykład 2: drukarka jako zasób niepodzielny



- dwa procesy chcą zmodyfikować współdzielony zasób

Wyścigi i sekcje krytyczne

Przykład 3: zmiana zmiennej współdzielonej

Dwa wątki współdzielą zmienną globalną typu całkowitego `n` i wykonują kod: `n++`, który oznacza

1. pobierz aktualną wartość `n` i umieść ją w rejestrze
2. dodaj 1 do wartości w rejestrze
3. zapisz nową wartość `n` do pamięci

Wątek A

pobierz n (17)

zwiększ wartość n (17->18)

zapisz n (18)

...

...

...

Wątek B

...

...

...

pobierz n (18)

zwiększ wartość n (18->19)

zapisz n (19)

Wątek A

pobierz n (17)

...

zwiększ wartość n (17->18)

...

zapisz n (18)

...

Wątek B

...

pobierz n (17)

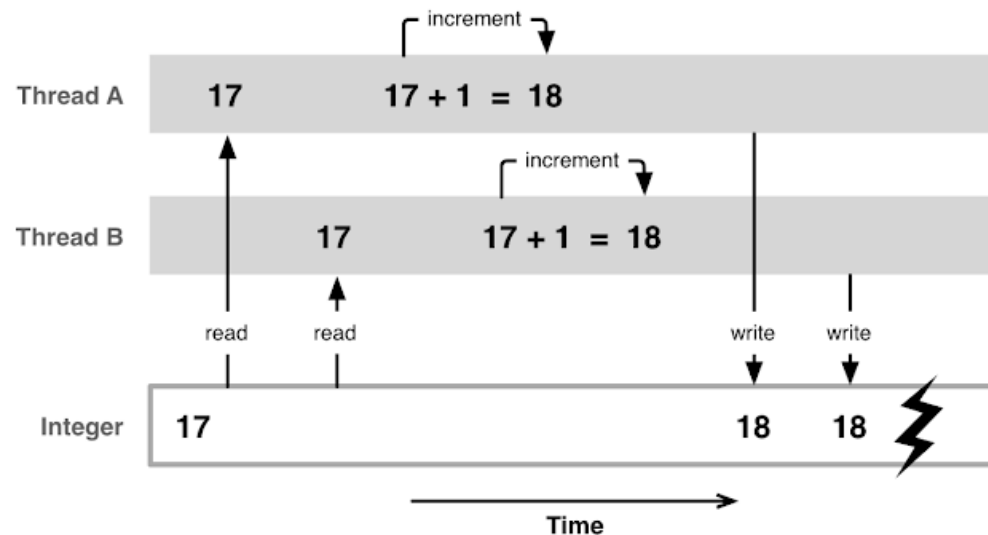
...

zwiększ wartość n (17->18)

...

zapisz n (18)

Wyścigi

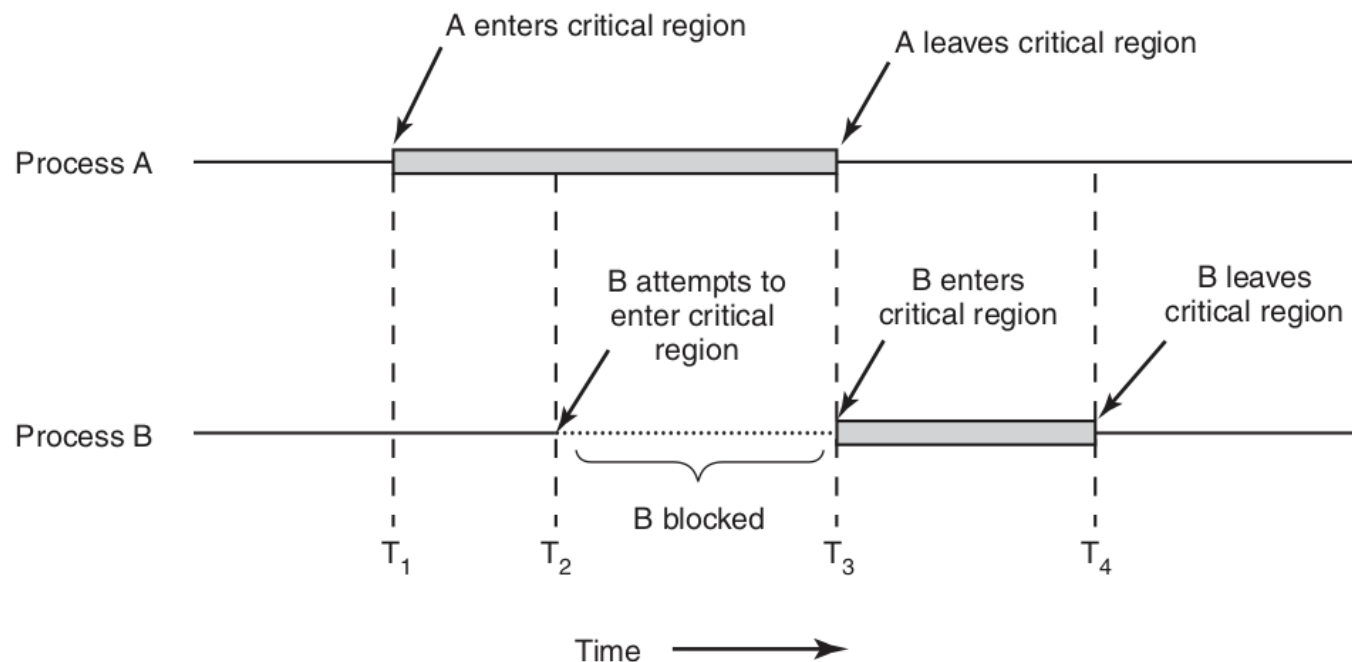


- zwiększanie wartości zmiennej globalnej powinno być wykonywane w sposób niepodzielny (atomowy)
- **operacje atomowe** (*atomic operation*) to niepodzielna operacja, która jest wykonywana w całości albo wcale, zapewnia poprawne wykonanie operacji na zasobach współdzielonych

Wyścigi i sekcje krytyczne

- wspólnie użytkowane zasoby wymagają ochrony przed współbieżnym dostępem, gdyż współbieżność wątków wykonania może prowadzić do powstanie niespójności w danych
- sytuacje, w których procesy (wątki) czytają oraz modyfikują pewne dzielone dane i rezultat końcowy zależy od tego, kiedy dokładnie każdy z tych procesów (wątków) będzie je czytał/modyfikował, nazywamy **wyścigiem** (*race condition*), hazardem lub przeplotem operacji.
- **sekcja krytyczna** (*critical section*) - część programu, w której następuje czytanie i modyfikowanie dzielonych danych
- aby zapobiec wyścigom musimy zapewnić **wzajemne wykluczenie**

Wzajemne wykluczenie



Wzajemne wykluczenie (*mutual exclusion*) polega na zapewnieniu takich warunków działania systemu, by tylko jeden proces mógł korzystać z zasobów niepodzielnych, tj. tylko jeden proces mógł przebywać w sekcji krytycznej

Wymogi dotyczące synchronizacji procesów

1. żadne dwa procesy nie mogą przebywać jednocześnie w swojej sekcji krytycznej
2. nie można czynić żadnych założeń, co do szybkości i liczby CPU
3. żaden proces wykonujący się poza sekcją krytyczną nie może blokować innych procesów
4. żaden proces nie może czekać w nieskończoność na wejście do swojej sekcji krytycznej.

Jak zapewnić wzajemne wykluczanie?

- jądro bez wywłaszczenia
- wyłączanie przerwań
- zmienne blokujące
- blokady wirujące (*spin lock*)
- semafony, muttaxy
- monitory

Wyłączanie przerwań

Proces wyłącza przerwania (także zegarowe), kiedy wchodzi do swojej sekcji krytycznej i włącza je, kiedy ją opuszcza.

Wady:

- proces użytkownika nie powinien mieć możliwości wyłączania przerwań, technika przydatna tylko gdy używane przez jądro
- w systemach wieloprocesorowych wyłączanie przerwań dotyczy tylko jednego procesora

Zalety:

- łatwy sposób modyfikowania struktur jądra w spójny sposób

Aktywne czekanie (*busy waiting*) - wątek oczekujący na zwolnienie zasobu pozostaje w pętli, stale sprawdzając warunek synchronizacji, aż zostanie spełniony. Jest to tzw. **blokada wirująca** (*spin lock*)

Wada: procesor jest w pełni obciążony wykonaniem pętli, może być nieefektywne w kontekście zużycia zasobów procesora

Wzajemne wykluczenie z aktywnym czekaniem

- zmienne blokujące (*lock variables*)
proste w użyciu ale nie rozwiązują problemu wyścigu, zmienne blokujące również są zasobem współdzielonym
- ściśła zmienność (*strict alternation*)
- algorytm Petersona
- instrukcja TSL (*Test and Set Lock*)

Ścisła zmienna

Procesy uzyskują dostęp naprzemiennie, wartość zmiennej `turn` określa, który proces może wejść do sekcji krytycznej

Początkowo `turn = 0`

proces A

```
-----  
while (TRUE) {  
    while (turn != 0) /* wait */;  
    critical_section();  
    turn = 1;  
    noncritical_section();  
}
```

proces B

```
-----  
while (TRUE) {  
    while (turn != 1) /* wait */;  
    critical_section();  
    turn = 0;  
    noncritical_section();  
}
```

Problem:

- gdy procesy różnią się szybkością, może dojść do blokowania przez proces wykonujący się poza sekcją krytyczną

Algorytm Petersona

```
#define FALSE 0
#define TRUE  1
#define N      2          /* number of processes */
int turn;                /* whose turn is it? */
int interested[N];        /* all values initially zero (FALSE) */

void enter_region (int process) /* process: who is entering (0 or 1) */
{
    int other;              /* number of the other process */
    other = 1 - process;    /* the opposite of process */
    interested[process] = TRUE; /* show that you are interested */
    turn = process;         /* set flag */
    while (turn == process && interested[other] == TRUE);
}

void leave_region(int process) /* who is leaving (0 or 1) */
    interested[process] == FALSE; /* indicate departure from critical region */
}
```

Instrukcja TSL

- rozkaz `tsl` kopiuje wartość zmiennej blokującej do rejestru i jednocześnie zmienia jej wartość, jest to gwarantowana sprzętowo **operacja niepodzielna**
- na czas wykonywania instrukcji procesor blokuje szynę pamięci aby żaden inny proces nie miał do niej dostępu

`enter_region:`

<code>tsl register,lock</code>	copy lock to register and set lock to 1
<code>cmp register,#0</code>	was lock zero?
<code>jne enter_region</code>	if it was not zero, lock was set, so loop
<code>ret</code>	return to caller; critical region entered

`leave_region:`

<code>mov lock,#0</code>	store a 0 in lock
<code>ret</code>	return to caller

Wymagana jest kooperacja procesów tak aby w odpowiednich momentach wołały procedury *enter_region* oraz *leave_region*.

Instrukcja TSL: zalety

- nadają się dla dowolnej liczby procesów (maszyny jedno- i wieloprocessorowe (SMP))
- są proste i łatwe w weryfikacji
- nadają się do kontroli wielu sekcji krytycznych

Instrukcja TSL: wady

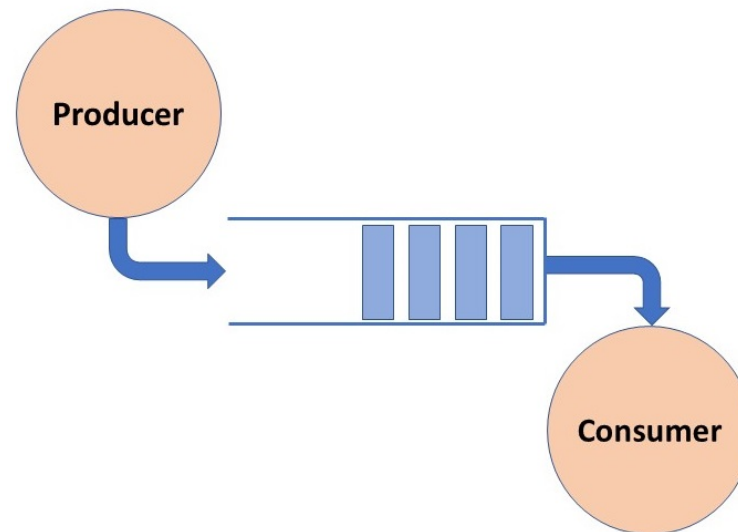
- aktywne czekanie marnuje czas CPU
- możliwość wystąpienia zagłodeń (*starvation*)
- możliwość wystąpienia zakleszczeń
- problem odwróconego priorytetu - proces aktywnie czekający o wyższym priorytecie, może nie dopuścić do przełączenia CPU na proces o niższym priorytecie przebywającym w sekcji krytycznej, powodując zakleszczenie

Wzajemne wykluczenie bez aktywnego czekania

Można uniknąć marnowania czasu jednostki centralnej na jałowe sprawdzanie warunku jeśli system udostępnia dwie operacje do synchronizacji pracy procesów: SLEEP oraz WAKEUP.

- SLEEP – odwołanie do systemu, które powoduje zablokowanie procesu wywołującego tę funkcję do czasu, aż nie nadejdzie sygnał budzenia
- WAKEUP – odwołanie do systemu służące do budzenia procesów

Problem producenta i konsumenta (ograniczonego bufora)



- proces producenta nie może tworzyć przedmiotu, jeśli bufor współdzielony jest pełny
- proces konsumenta nie może zużywać przedmiotu, jeśli współdzielony bufor jest pusty
- dostęp do współdzielonego bufora musi się wzajemnie wykluczać

Problem producenta i konsumenta: przykład

```
#define N      100                /* number of slots in the buffer */

int count = 0;                   /* number of items in the buffer */

void producer(void)
{
    int item;

    while (TRUE) {                /* repeat forever */
        produce_item(&item);      /* generate next item */
        if (count == N) sleep();  /* if buffer is full go to sleep */
        enter_item(item);         /* put item in buffer */
        count = count + 1;        /* increment number of items in buffer */
        if (count == N) wakeup(consumer); /* was buffer empty? */
    }
}
```

```
void consumer(void)
{
    int item;

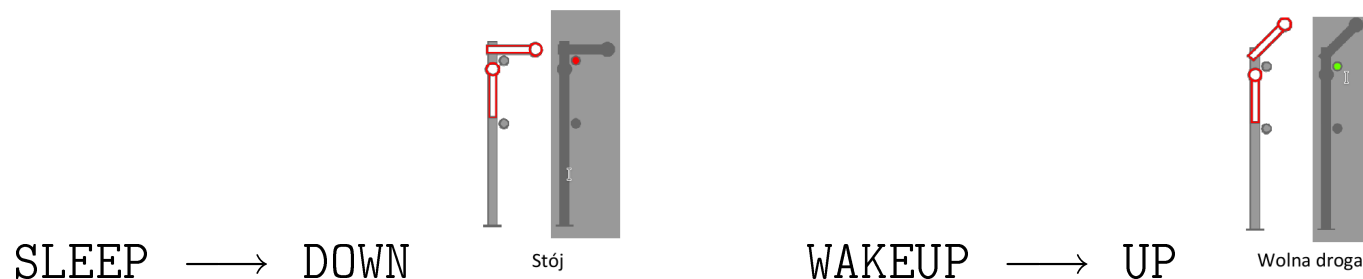
    while (TRUE) {                                /* repeat forever */
        if (count == 0) sleep();                  /* if buffer is empty go to sleep */
        remove_item(&item);                       /* take item out of buffer */
        count = count - 1;                        /* decrement number of items in buffer */
        if (count == N-1) wakeup(producer);       /* was buffer full? */
        consume_item(item);                       /* print item */
    }
}
```

Uwaga! Zmienna `count` odzwierciedlająca liczbę przedmiotów w buforze, nie jest chroniona, dostęp do niej nie jest niepodzielny. W przypadku wyścigu może dojść do zablokowania producenta i konsumenta

Semafor

Konieczność zapewnienia niepodzielności wykonywania operacji prymitywnych SLEEP i WAKEUP doprowadziła do powstania semaforów (E.W.Dijkstra, 1965).

Semafor to specjalna zmienna całkowita, na której są wykonywane operacje DOWN i UP²²:



Operacje UP i DOWN muszą być **niepodzielne** (atomowe), tj. w dowolnej chwili tylko jeden proces/wątek może je wykonywać.

²²Oryginalne nazwy operacji używane przez Dijkstra to P (*Proberen*, sprawdź) i V (*Vorhogen*, podnieś)

Semafor

Definicje operacji opuszczania (DOWN) i podnoszenia (UP) semafora:

```
void DOWN(semaphore) {  
    if (semaphore > 0 ) {  
        semaphore = semaphore - 1;  
        continue();  
    } else {  
        sleep(); /* add the process to semaphore queue and block it */  
    }  
}
```

```
void UP(semaphore) {  
    semaphore = semaphore + 1;  
    wakeup(); /* wakeup all blocked processes and make them runnable */  
              /* scheduler selects one to run */  
}
```

Problem producenta i konsumenta (rozwiązanie z semaforami)

```
#define N      100                                /* number of slots in the buffer */
typedef int semaphore;                             /* semaphores are a special kind of int */
semaphore mutex = 1;                               /* controls access to critical section */
semaphore empty = N;                               /* count empty buffer slots */
semaphore full  = 0;                               /* count full buffer slots */

void producer(void)
{
    int item;
    while (TRUE) {                                /* TRUE is the constant 1 */
        produce_item(&item);                       /* generate next item */
        down(&empty);                               /* decrement empty count */
        down(&mutex);                               /* enter critical region */
        enter_item(&item);                          /* put new item in buffer */
        up(&mutex);                                  /* leave critical region */
        up(&full);                                   /* increment count of full slots */
    }
}
```

```
void consumer(void)
{
    int item;
    while (TRUE) {
        down(&full);
        down(&mutex);
        remove_item(&item);
        up(&mutex);
        up(&empty);
        consume_item(&item)
    }
}
```

/* infinite loop */
/* decrement full count */
/* enter critical region */
/* take item from buffer */
/* leave critical area */
/* increment count of empty slots */
/* do sth with the item */

Przeznaczenie semaforów:

- semafor binarny otrzymuje początkową wartość 1 i jest używany przez dwa lub więcej procesów, aby zapewnić że tylko jeden z nich może w danej chwili przebywać w swojej sekcji krytycznej
 - mutex jest **semaforem binarnym**
 - mutex zapewnienia wzajemne wykluczanie się procesów (*MUTual EXclusion*)
- semafony empty oraz full służą do **synchronizacji**, zapewniają, że zdarzenia będą zachodziły we właściwej kolejności

Semafor i przerwania

Jak można wykorzystać semafor przy obsłudze przerwań?

- z każdym urządzeniem wej/wyj wiąże się semafor, którego początkowa wartość wynosi zero
- po zainicjowaniu operacji wej/wyj dla danego urządzenia proces wykonuje DOWN na semaforze i przechodzi w stan uśpienia
- po nadejściu przerwania proces obsługujący to przerwanie wykonuje UP na semaforze
- proces, który zgłosił żądanie wej/wyj staje się gotowy do wykonania

Wady semaforów

- semafor jest wysokopoziomową abstrakcją opartą na niskopoziomowych mechanizmach elementarnych, które zapewniają niepodzielność i dostarczają mechanizmów wstrzymywania
- wstrzymywanie i wznowianie wymaga przełączania kontekstu i zmian w kolejkach modułu szeregowania i kolejek wątków wstrzymanych; operacje na semaforze są **powolne**

Blokady pętlowe, wirujące (*spinlocks*)

- najprostszy elementarny mechanizm blokowania
- główna zaleta: niski koszt
- wątek próbujący pozyskać zasób, aktywnie oczekuje aż do odblokowania zasobu
- wątek aktywnie oczekuje na zasób na jednym procesorze, podczas gdy inny wątek korzysta z tego zasobu na innym procesorze
- wątki nie oddają procesora, jeśli nałożyły blokadę wirującą

blokady wirujące = blokady proste

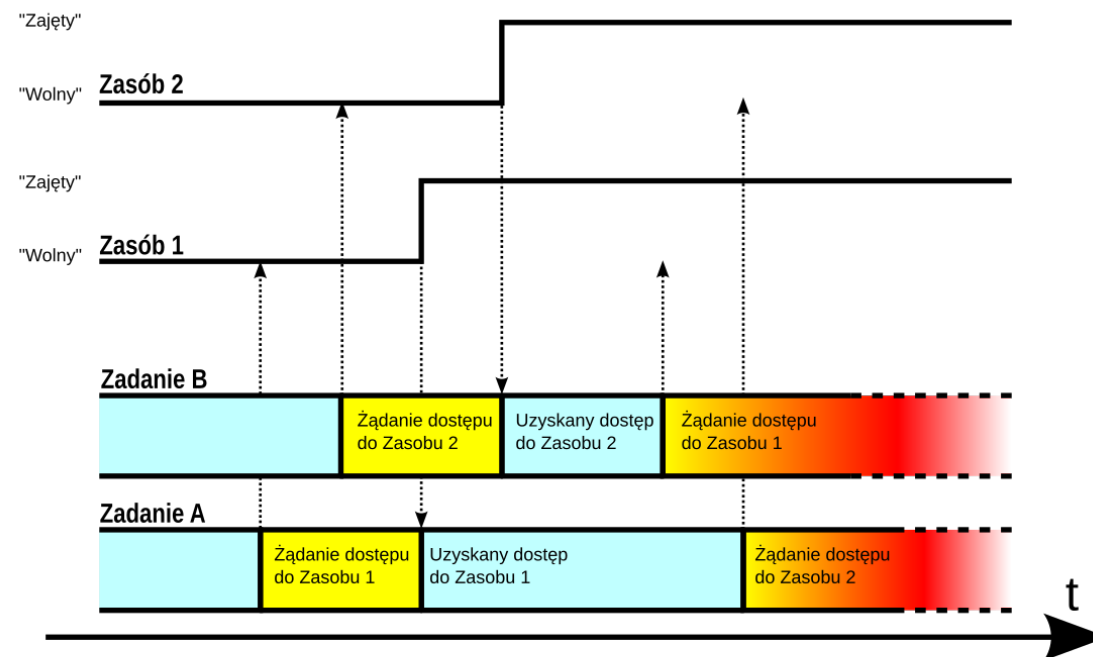
Monitory

Monitor to programistyczne narzędzie wymuszające wzajemne wykluczanie i koordynację procesów. Realizację struktury monitorów zawierają takie języki programowania jak Concurrent Pascal, Pascal-Plus, Modula, Java.

Monitorem nazywamy moduł programowy składający się z jednej lub kilku procedur, ciągu inicjującego i danych lokalnych o następujących cechach:

1. Zmienne danych lokalnych są dostępne wyłącznie za pośrednictwem procedur monitora (żadna procedura zewnętrzna nie może udostępniać zmiennych lokalnych).
2. Proces uruchamia monitor, wywołując jedną z jego procedur.
3. W tej samej chwili pod kontrolą monitora może działać tylko jeden proces; każdy inny proces, wywołujący w tym samym czasie monitor, musi zostać zawieszony do czasu zwolnienia monitora.

Zakleszczenia



Zakleszczenia (*deadlocks, deadly embraces*)

Zbiór procesów jest w stanie zakleszczenia, kiedy każdy z procesów ze zbioru czeka na zdarzenie, które może spowodować inny z procesów.

Zakleszczenie zdarza się, gdy każdy z procesów otrzymał wyłączone prawo do używania jakiegoś zasobu i dodatkowo żąda dostępu do zasobu już zajętego.

Przykłady:

- Proces A otrzymuje dostęp do drukarki, a proces B do dysku, a następnie proces A żąda dostępu do dysku, a proces B do drukarki.
- Proces A otrzymał prawo dostępu do dwóch (lub więcej) rekordów bazy danych, na których operacje mają być przeprowadzane w tym samym czasie przez proces B.

Zakleszczenia

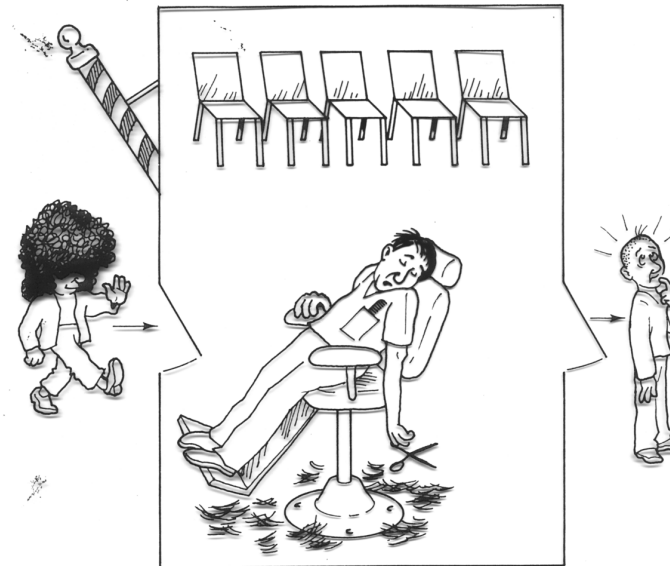
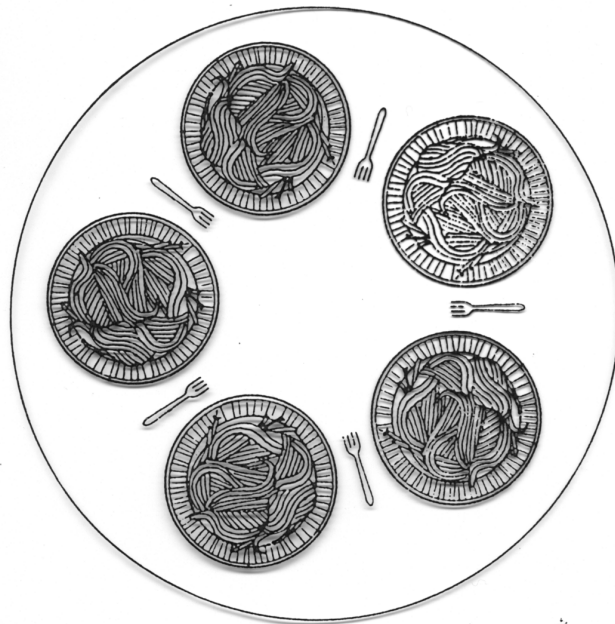
Jak unikać zakleszczeń?

1. Jeśli to tylko możliwe, to należy unikać stosowania blokad (zamiast blokad można stosować operacje atomowe lub używać struktur danych nie wymagających zakładania blokad).
2. Jeśli blokada jest niezbędna, to należy stosować blokadę pojedynczą.
3. Jeśli potrzebnych jest kilka blokad, to należy zdefiniować (małą) lokalną hierarchię blokad lub zastosować blokowanie stochastyczne.

Klasyczne problemy komunikacji międzyprocesowej

- problem producenta-konsumenta – modelowanie procesów synchronizacji między procesami (problem ograniczonego buforu)
- problem uczących filozofów – modelowania procesów, które współzawodniczą w dostępie do ograniczonych zasobów, np. urządzeń wej/wyj
- problem pisarzy i czytelników – zagadnienie zapewnienia prawidłowego dostępu do bazy danych (w trybie odczytu i zapisu)
- problem śpiącego fryzjera

Problemy uczujących filozofów i śpiącego fryzjera



Słownik skrótów

3DNow (*3D NO Waiting*) bez stanów oczekiwania 3D

AC (*accumulator*) akumulator

AGP (*Accelerated Graphics Port*) przyspieszony port graficzny, port akcelerowanej grafiki

AGU (*Address Generation Unit*) jednostka generowania adresów

ALU (*Arithmetic-Logic Unit*) jednostka arytmetyczno-logiczna

API (*Application Programming Interface*) interfejs programistyczny aplikacji

APIC (*Advanced Programmable Interrupt Controller*) zaawansowany programowalny kontroler przerwań

ATA (*AT Attachment*) końcówka AT

ASCII (*American Standard Code for Information Interchange*) amerykański standard kodowania na potrzeby wymiany informacji

BIOS (*Basic Input/Output System*) podstawowy system wejścia/wyjścia

BSD (*Berkeley Software Distribution*) wersja systemu uniksopodobnego z Berkeley

Btrfs (*B-tree File System*) system plików oparty o B-drzewo

CD-ROM (*Compact Disk Read-Only Memory*) pamięć tylko do odczytu na płycie kompaktowej

CHS (*Cylinder/Head/Sector*) cylinder/głowica/sektor

CFS (*Completely Fair Scheduler*) planista całkowicie sprawiedliwy

CIFS (*Common Internet File System*) powszechny internetowy system plików

CISC (*Complex Instruction Set Computer*) komputer o pełnej liście rozkazów

CLI (*Command Line Interface*) interfejs wiersza poleceń

CMOS (*Complementary Metal-Oxide Semiconductor*) komplementarny półprzewodnik tlenkowy

COW (*Copy On Write*) kopiowanie przy zapisie

CPU (*Central Processing Unit*) jednostka centralna

CRC (*cyclic redundancy check*) cykliczna kontrola nadmiarowa

DAC (*discretionary access control*) uznaniowa kontrola dostępu

DMA (*Direct Memory Access*) bezpośredni dostęp do pamięci

DMI (*Direct Media Interface*) bezpośredni interfejs multimedialny, magistrala łącząca mostek północny z południowym

DRAM (*Dynamic RAM*) dynamiczna pamięć RAM

EDVAC (*Electronic Discrete Variable Computer*) elektroniczny komputer automatyczny o dyskretnych zmiennych

EEPROM (*Electrically Erasable and Programmable ROM*) elektrycznie wymazywalna i programowalna pamięć stała

EIDE (*Extended Integrated Device Electronics*) ulepszona zintegrowana elektronika napędów

EISA (*Extended Industry Standard Architecture*) rozszerzona standardowa architektura przemysłowa

EMS (*Expanded Memory Specification*) specyfikacja rozszerzonej pamięci

ENIAC (*Electronic Numerical Integrator and Computer*) elektroniczne urządzenie numeryczne całkujące i liczące

EPROM (*Erasable and Programmable ROM*) wymazywalna i programowalna pamięć stała

EXT2 (*Second Extended File System*) drugi rozszerzony system plików

- EXT3** (*Third Extended File System*) trzeci rozszerzony system plików
- EXT4** (*Fourth Extended File System*) czwarty rozszerzony system plików
- FAT** (*File Allocation Table*) tablica alokacji plików
- FCFS** (*First-Come, First-Served*) pierwszy nadszedł, pierwszy obsłużony
- FDD** (*Floppy Disk Drive*) stacja dyskietek
- FLOPS** (*Floating-point operations per second*) liczba operacji zmiennopozycyjnych na sekundę
- FIFO** (*First In, First Out*) pierwszy nadszedł, pierwszy obsłużony
- FSB** (*Front System Bus*) przednia magistrala systemowa
- FSF** (*Free Software Foundation*) Fundacja Wolnego Oprogramowania
- GDT** (*Global Descriptor Table*) globalna tablica deskryptorów
- GNU** (*Gnu is Not Unix*) Gnu nie jest Uniksem
- GPL** (*General Public Licence*) Ogólna Licencja Publiczna
- GPU** (*Graphics Processing Unit*) procesor graficzny
- GUI** (*Graphical User Interface*) graficzny interfejs użytkownika
- HDD** (*Hard Disk Drive*) dysk twardy
- HPC** (*High-Performance Computing*) wysokowydajne obliczenia

- HT** (*Hyper-Threading*) hiperwątkowość
- IBR** (*instruction buffer register*) rejestr bufora rozkazów
- IDT** (*Interrupt Descriptor Table*) tablica deskryptorów przerwań
- IEU** (*Integer Execution Unit*) jednostka wykonawcza dla liczb stałopozycyjnych
- I/O** (*Input/Output*) wejście-wyjście
- IPC** (*InterProcess Communication*) komunikacja międzyprocesowa
- IPI** (*Inter-Processor Interrupt*) przerwanie międzyprocesorowe
- IR** (*instruction register*) rejestr rozkazów
- IRQ** (*Interrupt ReQuest*) linia/żądanie przerwania
- ISA** (*Instruction Set Architecture*) architektura zestawu instrukcji (model programowy procesora)
- ISR** (*Interrupt Service Routine*) procedura obsługi przerwania
- JCL** (*Job Control Language*) język kontroli zadań
- JFS** (*Journaling File System*) system plików z kroniką
- KVM** (*Kernel-based Virtual Machine*) maszyna wirtualna realizowana przez jądro systemu operacyjnego

- LBA** (*Linear Block Addressing*) liniowa adresacja bloków
- LDT** (*Local Descriptor Table*) lokalna tablica deskryptorów
- LSI** (*Large Scale Integration*) duży stopień scalenia
- LWP** (*Light Weight Process*) lekki proces
- LVM** (*Logical Volume Manager*) menadżer woluminów logicznych
- MAC** (*Mandatory Access Control*) obowiązkowa kontrola dostępu
- MAR** (*Memory Address Register*) rejestr adresowy pamięci
- MBR** (*Memory Buffer Register*) rejestr buforowy pamięci
- MBR** (*Master Boot Record*) główny rekord rozruchowy
- MQ** (*Multiplier-Quotier*) rejestr mnożenia i dzielenia
- MIMD** (*Multiple Instruction Multiple Data*) wiele instrukcji wiele danych
- MMIO** (*Memory-Mapped Input/Output*) wejście/wyjście mapowane w pamięci
- MMX** (*MultiMedia Extensions*) rozszerzenia multimedialne
- MPP** (*Massively Parallel Processing*) przetwarzanie masywnie równoległe
- MS-DOS** (*Microsoft Disk Operating System*) dyskowy system operacyjny
Microsoft

- MTA** (*Mail Transport Agent*) agent transportu poczty
- MTBF** (*Mean Time Between Failures*) średni czas międzyawaryjny
- NMI** (*NonMaskable Interrupts*) przerwania niemaskowalne
- NVRAM** (*Non-Volatile RAM*) nieulotna pamięć RAM
- NTFS** (*New Technology File System*) system plików nowej technologii
- NUMA** (*Non-Uniform Memory Access*) niejednolity dostęp do pamięci
- NVRAM** (*Non-Volatile RAM*) nieulotna pamięć RAM
- PATA** (*Parallel AT Attachment*) równoległa końcówka AT
- PC** (*Program Counter*) licznik programu
- PC** (*Personal Computer*) komputer osobisty
- PCB** (*Process Control Block*) blok kontrolny procesu
- PCI** (*Peripheral Component Interconnect*) połączenie komponentów peryferyjnych
- PIC** (*Programmable Interrupt Controller*) programowalny kontroler przerwań
- PnP** (*Plug and Play*) podłącz i używaj

POSIX (*Portable Operating System Interface for UNIX*) przenośny interfejs systemu operacyjnego dla Uniksa

PROM (*Programmable ROM*) programowalna pamięć stała

RAID (*Redundant Array of Inexpensive Disks*) nadmiarowa macierz niedrogich dysków lub (*Redundant Array of Independent Disks*) nadmiarowa macierz niezależnych dysków

PSW (*Program Status Word*) słowo stanu programu

RAM (*Random Access Memory*) pamięć swobodnego dostępu

RISC (*Reduced Instruction Set Computer*) komputer o zredukowanej liście rozkazów

ROM (*Read-Only Memory*) pamięć stała/tylko do odczytu

RPC (*Remote Procedure Call*) zdalne wywołanie procedury

SATA (*Serial AT Attachment*) szeregową końcówka AT

SAS (*Serial Attached SCSI*) szeregowy interfejs SCSI

SCSI (*Small Computer System Interface*) interfejs małych systemów komputerowych

SIMD (*Single Instruction Multiple Data*) pojedyncza instrukcja wiele danych

SJF (*Shortest Job First*) najkrótsze zadanie najpierw

SMB (*Server Message Block*) blok komunikatów serwera

SMP (*Symmetric MultiProcessing*) symetryczne przetwarzanie wieloprocessorowe

SPN (*Shortest Process Next*) najkrótszy proces następny

SPOOL (*Simultaneous Peripheral Operations On-Line*) jednoczesna bezpośrednia praca urządzeń

SRAM (*Static RAM*) statyczna pamięć RAM

SRT (*Shortest Remaining Time*) najkrótszy pozostały czas

SSD (*Solid State Device*) dysk półprzewodnikowy

SSE (*Streaming SIMD Extensions*) strumieniowe rozszerzenia SIMD

TLB (*Translation Lookaside Buffer*) bufor translacji adresów

TSL (*Test and set Lock*) testuj i załóż rygiel

TSS (*Task Status Segment*) segment statusu zadania

USB (*Universal Serial Bus*) uniwersalna szyna szeregową

VFS (*Virtual File System*) wirtualny system plików

VLSI (*Very Large Scale Integration*) bardzo duża skala integracji