# Wirtualizacja

a) **Pełna wirtualizacja** – binarna translacja instrukcji z systemu gościa do sprzętu poprzez system hosta:
   i) Wirtualizowany system operacyjny nie wymaga żadnych zmian/modyfikacji.
   ii) Większe bezpieczeństwo w izolacji awarii.
b) **Parawirtualizacja** – część instrukcji i odwołań systemu gościa jest tłumaczonych, a część jest przekazywana natywnie do sprzętu:
   i) Efektywniejsze wykorzystanie współdzielonych zasobów.
   ii) Mniejszy narzut obliczeniowy na hosta.
c) **Wirtualizacja typu 1** - całość instrukcji jest przekazywana natywnie do sprzętu
   i) Najbardziej efektywne rozwiązanie

Przydzielanie zasobów:

1) *CPU* - można w sumie więcej niż ogólnie dostępne
2) *RAM*
3) *DYSK* - można w sumie więcej niż ogólnie dostępne (niestety są duże czasy dostępu)

**Wirtualizacja sprzętowa** - obecnie najpopularniej używany sposób wirtualizacji (*VT-x* i *AMD-V*)

# Uprawnienia do plików - zastosowania

| | | |
|---|---|---|
| 0 | --- | blokada |
| 1 | --x | katalog bez podglądu |
| 2 | -w- | zbieranie sekretnych logów |
| 3 | -wx | nieprzydatne |
| 4 | r-- | stała konfiguracja |
| 5 | r-x | pliki wykonywalne, katalogi |
| 6 | rw- | pliki edytowalne |
| 7 | rwx | skrypty i katalogi usera |

| | | | |
|---|---|---|---|
| | ??s | bit suid | programy specjalne *(program uruchomiony z uprawnieniami właściciela)* |
| | ??t | sticky bit | katalog specjalny *(użytkownicy mogą tworzyć w nim pliki, nie mogą za to usuwać nie swoich plików)* |

*Uprawnienia na temat usuwania plików zależą od uprawnień katalogu.*

## System operacyjny

zarządza sprzętem komputerowym, jest pośrednikiem między użytkownikiem, a sprzętem komputerowym

**Sprzęt komputerowy** - CPU, RAM, I/O devices

**Zadania systemu operacyjnego:**

a) dostarczenie środowiska do uruchamiania i zarządzania (**control program**) programami użytkownika (wygoda),
   i) *Nadzorowanie działania programów użytkowych.*
   ii) *Przechwytywanie i przeciwdziałanie błędom.*
   iii) *Udostępnianie systemu komputerowego użytkownikom. (tworzenie podsystemów)*
   iv) *Kontrola dostępu użytkowników i programów do zasobów.*
   v) *Obsługa i kontrola pracy urządzeń wejścia-wyjścia.*

b) dystrybucja zasobów (**resource allocator**) do efektywnej eksploatacji sprzętu komputerowego (wydajność).
   i) Planowanie i przydział czasu procesora. (zapewnia płynność działania systemu)
   ii) Kontrola i przydział pamięci operacyjnej.
   iii) Zarządzanie pozostałymi zasobami, jak oprogramowanie czy dostęp do sieci internet.
   iv) Dostarczenie mechanizmów do synchronizacji zadań i komunikacji między zadaniami.

<table>
<tr><td colspan="2">Użytkownicy</td></tr>
<tr><td colspan="2">Programy użytkowe</td></tr>
<tr><td rowspan="2">System operacyjny</td><td>Powłoka</td></tr>
<tr><td>Jądro i moduły</td></tr>
<tr><td colspan="2">BIOS - Basic Input Output System</td></tr>
<tr><td colspan="2">Sprzęt komputerowy</td></tr>
</table>

**Podział systemów komputerowych:**

a) ze względu na wielkość
   i) *Mainframe (może oznaczać połączenie komputerów tzw. cluster)*
   ii) *Minicomputer*
b) ze względu na zasoby (oprogramowanie)
   i) *server*
   ii) *workstation*

**System operacyjny to jedyny program działający cały czas na komputerze! (jądro/kernel)**

Pozostałe typy programów:

a) systemowe (np. polecenia systemowe)
b) aplikacyjne (aplikacje użytkowe)

Jednostki danych

a) bit (*b*) - ma wartość 1 lub 0
   i) 1 Kib - 1024 b
   ii) 1 Mib - 2^20 b
   iii) 1 Gib - 2^30 b
b) bajt (*B*) - 8 bitów
   i) 1 KB - 1024 B
   ii) 1 MB - 2^20 B
   iii) 1 GB - 2^30 B
c) word - natywna jednostka dla danej architektury (np. dla 64-bitowej słowo to 8 bajtów)

Elementy systemu komputerowego

a) **Sprzęt**: procesor CPU, pamięć RAM, urządzenia we-wy
b) **Systemy operacyjne**: Linux/Unix, Windows, MacOS, itp.
c) **Programy użytkowe**: aplikacje, systemy baz danych, gry komputerowe, oprogramowanie biurowe, środowiska programistyczne, itp.
d) **Użytkownicy**: ludzie, programy, maszyny

**Uruchamianie systemu operacyjnego:**
ładowanie BIOS i Bootstrapu z ROM'u -> przekazanie sterowania do CPU -> ładowanie systemu operacyjnego -> załadowanie procesu "init"

Przerwania pozwalają na zsynchronizowanie pracy zasobów o różnej wydajności, sygnalizują one wystąpienie zdarzenia.

a) Sprzętowe przerwanie może być wyzwolone przez przesłanie sygnału przez szynę systemową do procesora
b) Programowe przerwanie może być wyzwolone specjalną operacją, tzw. system call lub monitor call

W momencie wystąpienia przerwania, procesor przerywa aktualnie wykonywaną operację, wykonuje procedurę przewidzianą dla danego zdarzenia i wraca do przerwanej operacji.

## Struktura pamięci

| | | |
|---|---|---|
| **RAM** | *random access memory* | pamięć, do której ładowane są programy do wykonania oraz dane dla tych programów |
| **DRAM** | *dynamic random-access memory* | technologia półprzewodnikowa z której korzysta RAM |
| **ROM** | *read-only memory* | pamięć, w której przechowywane są niezmienialne programy, np. wgrane gry do konsol wideo |
| **EEPROM** | *erasable programmable read-only memory* | pamięć, która nie może być zbyt często nadpisywana, np. system operacyjny urządzeń mobilnych |
| **Rejestr** | | szybka, niewielka pamięć przy/w procesorze na czas wykonywania pojedynczej instrukcji |

| | | |
|---|---|---|
| **Pamięć NVRAM** | *nonvolatile RAM* | odpowiedni pamięci RAM podtrzymywany bateryjnie |
| **Pamięć stała (np. HDD)** | | dysk optyczny, pen drive, itp. |

*Pamięć ROM zachowuje dane po odcięciu napięcia, odwrotnie niż RAM.*

# Historia rozwoju systemów operacyjnych

1) **Proste systemy wsadowe:**
   a) Pierwsze komputery:
      i) *Wejście*: czytniki kart i przewijaki taśm
      ii) *Wyjście*: drukarki wierszowe, przewijaki taśm, perforatory kart
   b) Zadanie na karcie perforowanej: program, dane, karty sterujące
   c) Czas obliczeń: > minuty (czasem dni)
   d) System operacyjny umieszczony na stałe w pamięci operacyjnej
   e) Grupowanie zadań o podobnych wymaganiach: **wsad** (batch)
   f) Komputer obsługiwał operator, który pobierał i sortował programy
   g) Istotna różnica w szybkości działania procesora w porównaniu z we/wy
   h) Następstwem było wprowadzenie technologii dyskowej

2) **Wieloprogramowe systemy wsadowe:**
   a) Zastosowanie pamięci o dostępie swobodnym (dysków)
   b) Wczytywanie kart na dysk i zapamiętanie położenia danych => spooling
   c) Pula zadań (job pool) - wczytanie pewnej liczby zadań na dysk
   d) Możliwość dobierania zadań z dysku tak, aby zwiększyć efektywność jednostki centralnej
   e) Planowanie zadań (scheduling) - planowanie zadań i planowanie przydziału procesora

3) **Systemy z podziałem czasu:**
   a) Problemy systemów wieloprogramowych:
      i) wielowariantowość ścieżek wykonywania
      ii) brak możliwości modyfikowania programu

iii) długi czas od rozpoczęcia tworzenia programu do wyniku jego działania

b) Podział czasu, wielozadaniowość, multitasking
c) Interakcyjność, hands-on - wymiana danych z programem w ciągu jego trwania
d) System plików (file, filesystem):
   i) zestaw powiązanych informacji
   ii) format, typ
   iii) organizacja w katalogi
e) Bezpośredni dostęp użytkownika do komputera (bez operatora)
f) Pamięć wirtualna - wspomaganie pamięci operacyjnej pamięcią dyskową
g) Problem zakleszczenia - wzajemne oczekiwanie programów na zasoby

**4) Systemy dla komputerów osobistych:**

a) Zmniejszenie cen sprzętu
b) Rozwój linii komputerów PC (personal computer)
c) Początkowo systemy operacyjne dla pierwszych komputerów osobistych: nie wielostanowiskowe, nie wielozadaniowe, lecz z czasem je rozwinięto
d) **Microsoft:** MS-DOS, później Microsoft Windows
e) **IBM:** MS-DOS (od Microsoft), później OS/2
f) **Apple:** Macintosh, później iOS
g) **Bell Laboratories:** UNIX dla PDP-11 (wiele koncepcji z systemu MULTICS dla komputera GE 645)
h) Na bazie rozwiązań UNIX w latach ‘80 => Windows NT, IBM OS/2, Macintosh Operating System

**5) Systemy wieloprocesorowe, równoległe:**

a) Zwiększona przepustowość
b) Współczynnik przyspieszenia, który nie zawsze powiela wydajność
c) Współużytkowanie urządzeń zewnętrznych
d) Zwiększenie niezawodności (redundancja/nadmiarowość węzłów obliczeniowych)
e) Wieloprzetwarzanie symetryczne - w każdym procesorze działa identyczna kopia
   i) Działa *N* procesów na *N* egzemplarzach jednostki centralnej
   ii) Może się zdarzyć niezbalansowanie obciążenia procesorów
   iii) Wersja Encore systemu UNIX dla komputera

Multimax
   iv) SunOS w wersji 5 (Solaris 2) dla komputerów Sun
f) Wieloprzetwarzanie asymetryczne - każdy procesor ma inne zadanie (+procesor główny)
   i) Np. słabsze procesory obsługują komunikację (front-end)
   ii) IBM i komputer IBM Series/1 jako procesor czołowy
   iii) SunOS w wersji 4 dla komputerów Sun

**6) Systemy rozproszone, systemy klastrowe:**

a) Systemy luźno powiązane (loosely coupled), rozproszone (distributed systems)
b) Rozdzielenie geograficzne
c) Połączenie przez linie telekomunikacyjne (internet, linie telefoniczne, itp.)
d) Zróżnicowanie architektur poszczególnych węzłów (node)
e) Zróżnicowanie mocy obliczeniowych poszczególnych węzłów
f) **Cechy:**
   i) Podział zasobów
   ii) Przyspieszenie obliczeń
   iii) Niezawodność
   iv) Komunikacja

**7) Systemy czasu rzeczywistego (real-time):**

a) **Zastosowanie:**
   i) surowe wymagania na czas wykonania operacji lub przepływu danych
b) **Przykłady:**
   i) Jednokierunkowe sterowanie maszyną według zadanego programu
   ii) Odczytywanie wartości czujników
   iii) Analiza odczytanych wartości czujników i adekwatna reakcja robota
   iv) Analiza otoczenia, przetwarzanie danych (sygnałów, obrazów) i podejmowanie decyzji
c) **Odmiany systemów czasu rzeczywistego:**
   i) Rygorystyczny (*Hard real-time system*) - terminowe wypełnienie krytycznych zadań
   ii) Łagodny (*Soft real-time system*) - krytyczne zadanie do obsługi otrzymuje pierwszeństwo

System operacyjny musi tak zarządzać dostępem do pamięci, aby zminimalizować efekt NUMA.

a) **UMA** - *uniform memory access* - pamięć o jednorodnym czasie dostępu
b) **NUMA** - *non-uniform memory access* - pamięć o niejednorodnym czasie dostępu

| Tryby pracy systemu operacyjnego | |
|---|---|
| **Tryb użytkownika** (*mode* bit 1) | **Tryb jądra** (*mode* bit 0) |
| Procesy w trybie użytkownika mają ograniczone uprawnienia, co zapewnia większe bezpieczeństwo. Nie mają bezpośredniego dostępu do sprzętu ani pełnej kontroli nad pamięcią systemową. | Procesy w trybie jądra mają pełny dostęp do zasobów sprzętowych i pamięci systemowej. Mają nieograniczone uprawnienia, co umożliwia bezpośredni dostęp do sprzętu i zarządzanie pamięcią fizyczną. |

**Interfejs użytkownika**

a) Graficzny interfejs użytkownika (**GUI** - *graphical user interface*) to najpopularniejszy wśród użytkowników biurowych interfejs człowiek-komputer.
b) Interpreter poleceń (**CLI** - *command-line interface*) jest interfejsem użytkownika (**UI**) przyjmującym komendy tekstowe oraz funkcje.
c) Interfejs batch to zbiór komend i dyrektyw kontrolujących te komendy zapisanych w pliku, który po uruchomieniu kolejno uruchamia komendy z tego pliku.
d) Interpreter poleceń (*shell*): **sh** (*Bourne shell*), **bash** (*Bourne-Again shell*), **csh** (*C shell*), **zsh** (*Z shell*), **ksh** (*Korn shell*), itp.
e) Interpretacja poleceń: polecenie systemowe (np. *cd*) lub program (np. *rm*).

## Procesy

Tworzenie programu:

Kod źródłowy -> Kompilacja -> Linkowanie -> Kod binarny

Utworzenie procesu:

Kod binarny -> Wczytywanie do pamięci operacyjnej -> Nadanie PCB -> Proces

**Proces** to:
a) wczytany do pamięci i uruchomiony kod
b) jednostka pracy w systemie z dzieleniem czasu

Pierwsze systemy operacyjne: jeden program, który ma dostęp do wszystkich zasobów (tzw. *job*).
Obecnie: zarządzanie uruchomionymi programami, czyli procesami (*user program*, *task* obecnie *process*)

Zarządzanie procesami obejmuje: **kontrole** i **separacje**
System zawiera kolekcje procesów:
i) procesy systemu operacyjnego
ii) procesy użytkownika
Potencjalnie wszystkie w/w procesy uruchamiane są jednocześnie. Procesor(y) przełączają się między procesami, co zwiększa efektywność systemu komputerowego

| | |
|---|---|
| **Powoływanie** (tworzenie) | - nowe zadanie wsadowe<br>- interaktywne logowanie<br>- utworzenie usług przez system operacyjny<br>- podział (*spawn*) istniejącego procesu |
| **Terminowanie** | - prawidłowe zakończenie<br>- brak pamięci<br>- naruszenie ochrony<br>- interwencja operatora lub systemu operacyjnego |

Proces zostaje zatrzymany przez:
a) instrukcję HALT
b) akcję użytkownika
c) awarię lub błąd
d) terminowanie procesu rodzica

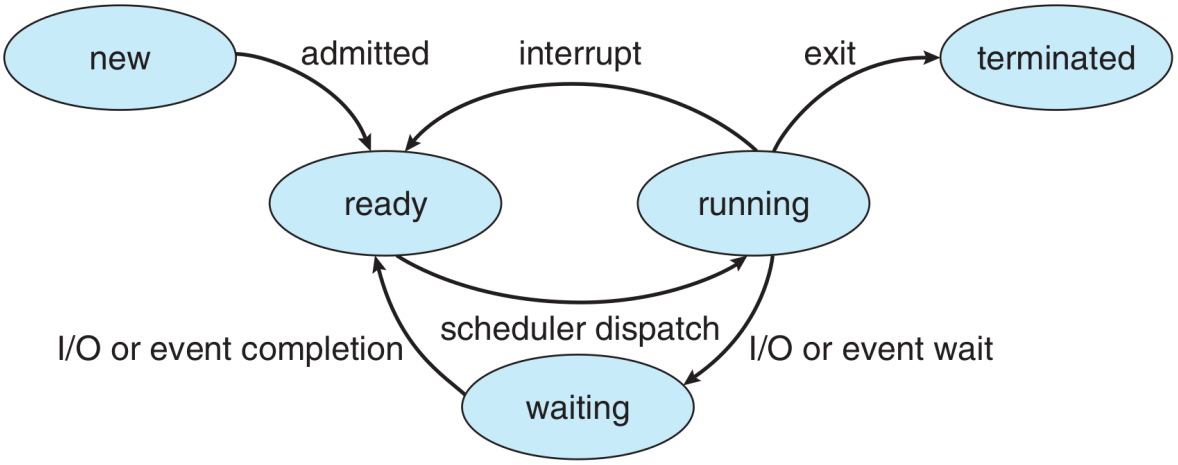
Powiązane komendy: `$ jobs`, `$ bg`, `$ fg`

Proces składa się z:

| |
|---|
| **Stos** (*stack*) - parametry procedur, adresy powrotne (po wykonaniu danej instrukcji), zmienne tymczasowe |
| ↓<br>↑ |
| **Sterta** (*heap*) - pamięć przydzielana dynamicznie |
| **Sekcja danych** (*data section*) - zmienne globalne (w **C** podzielne są na *zainicjalizowane* i *niezainicjalizowane*) |
| **Licznik programu** (*program counter*) |
| **Kod programu** (*text section*) |

Proces może być:

a) aktywny - licznik rozkazów wskazuje następną instrukcję
b) pasywny - plik wykonywalny zawiera listę instrukcji




Źródło: A. Silberschatz, Operating Systems Concepts Essentials

Powody zawieszenia procesu:

a) **Swapping** - system musi zwolnić pewną ilość pamięci, aby uruchomić proces, który jest gotowy do wykonania
b) **Inny problem SO** - błędy
**c) Interaktywne żądanie użytkownika**
d) **Timing** - periodyczne wywołanie procesu (np. monitorowanie) i może on zostać zawieszony do następnego wywołania
e) **Żądanie procesu macierzystego** - *SIGSTOP* / *SIGTSTP* / *SIGCONT* (SIGTSTP może zostać zignorowany SIGSTOP nie)

# PCB (process control block) - blok kontrolny procesu

Obszar pamięci zawierające różne informacje skojarzone z procesem, którego dotyczy. Jest rezerwowane przez jądro.

| Process state |
| --- |
| Process number |
| Program counter |
| Register |
| Memory limits |
| List of open files |
| . . . |

**Stan procesu:**

Polecenia:

```
$ ps -p pid_procesu -o pid,status,comm
$ cat /proc/pid_procesu/status | grep Stat
```

Możliwe stany:

a) R - running
b) S - sleeping in an interruptible wait
c) D - waiting in uninterruptible disk sleep
d) Z - zombie
e) T - traced or stopped (on a signal)
f) W - paging

**Numer procesu (*PID*):**

Polecenia:

```
$ pidof nazwa_programu
$ ps aux | grep nazwa_programu
```

Przydatne funkcje (*unistd.h*):

```
pid_t getpid(void);
pid_t getppid(void);
pid_t fork(void);
```

**Licznik rozkazów:**
Adres kolejnej instrukcji do wykonania.

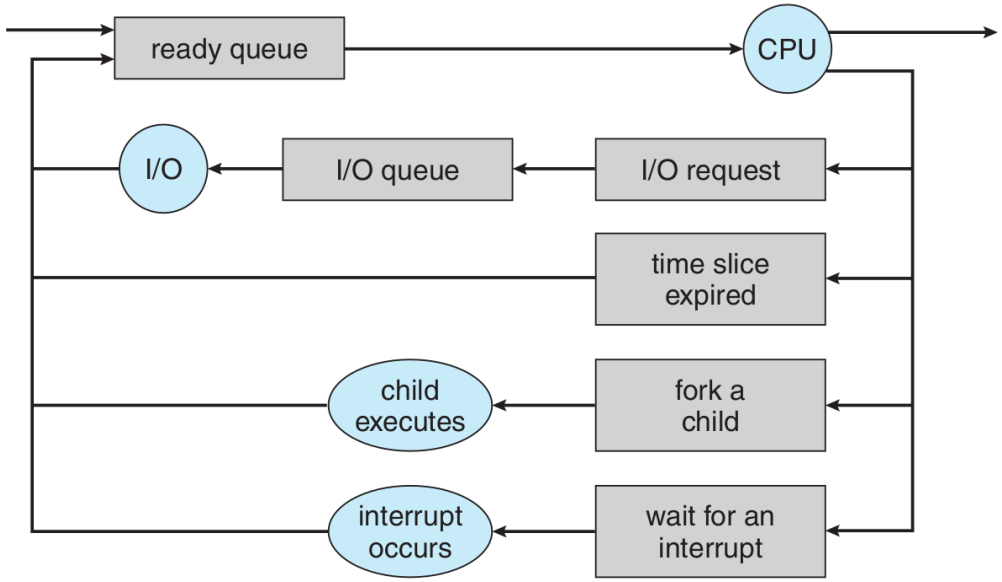
**Rejestry procesora (*CPU registers*):**
M.in. akumulatory, rejestry indeksowe, wskaźniki stosu, rejestry ogólnego przeznaczenia, rejestry warunków, etc.

**Informacje o planowaniu przydziału procesora (*CPU-scheduling information*):**
M.in. priorytet procesu, wskaźnik do kolejek, etc.

**Informacje o zarządzaniu pamięcią (*Memory-management information*):**
M.in. zawartość rejestrów granicznych, tablice stron lub tablice segmentów, etc.

**Informacje do rozliczeń (*Accounting information*):**
Ilość zużytego czasu procesora i czasu rzeczywistego, ograniczenia czasowe, numery kont, numer zadania lub procesu, itp.

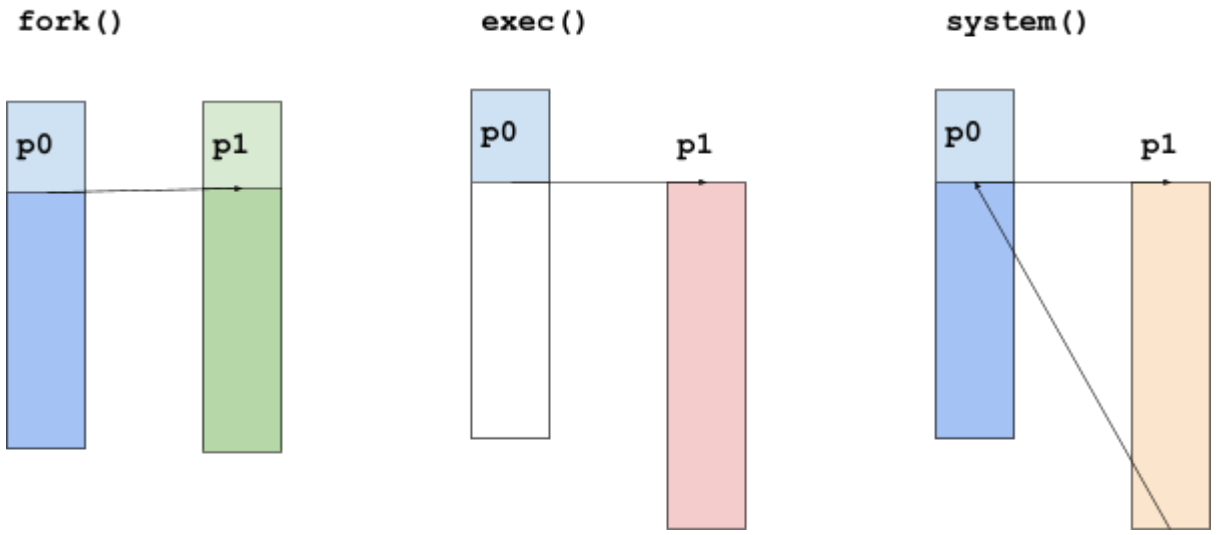
**Informacja o stanie wejścia-wyjścia (*I/O status information*):**
M.in. lista urządzeń we/wy przydzielonych do procesu, lista otwartych plików, itd.




Źródło: A. Silberschatz, Operating Systems Concepts Essentials

# Tworzenie procesów

a) Dany proces (*rodzic - parent*) może utworzyć wiele nowych procesów (*dziecko - child*)
b) Każdy z nowo powstałych procesów może tworzyć kolejne, powstaje drzewo - tree procesów
c) Identyfikacja procesu w większości systemów odbywa się przez identyfikator PID
d) Kiedy dany proces utworzy proces potomny, mogą zaistnieć dwa przypadki:
   i) Proces rodzica wykonuje się nadal jednocześnie z procesem potomnym (**fork**)
   ii) Proces rodzica czeka, aż któryś lub wszystkie procesy potomne zakończą działanie (**fork** lub **system**)
e) Są także dwie możliwości adresowania pamięci dla nowego procesu:
   i) Proces potomny jest duplikatem procesu rodzica, tzn. ma ten sam kod programu jak rodzic (**fork**)
   ii) Proces potomny to na nowo załadowany program (**exec** i **system**)



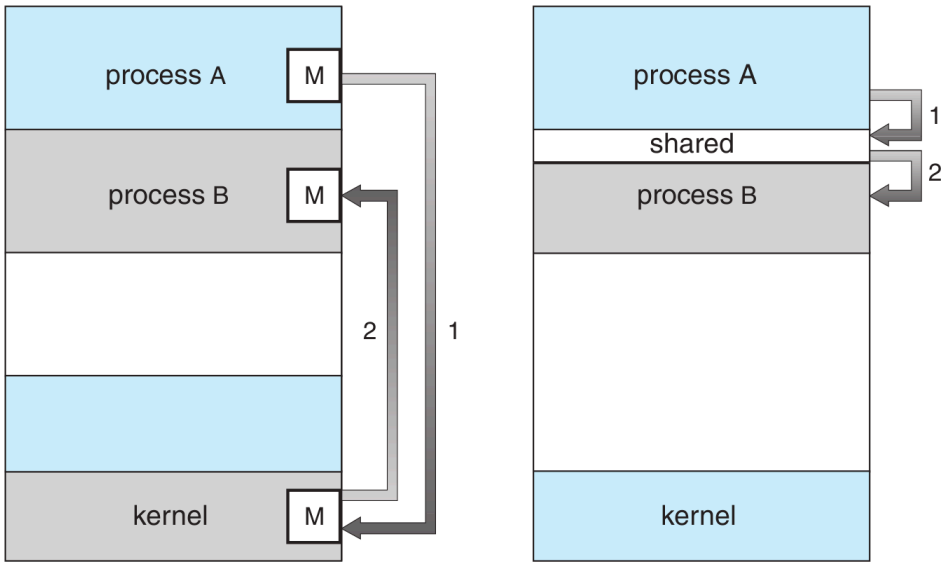

*Aby uruchomić nowy proces można wykorzystać połączenie funkcji fork() i exec().*

**Współistnienie procesów (*IPC - interprocess communication*)**

a) Proces niezależny - nie wpływają na niego, ani nie wpływa na inne procesy
b) Proces kooperujący - może wpływać na inne procesy lub inne procesy mogą wpływać na niego

**Znaczenie komunikacji międzyprocesowej:**

a) Współdzielenie informacji (dane, wymiana komunikatów)
b) Przyspieszenie obliczeń
c) Modularność systemu
d) Wygoda




Źródło: A. Silberschatz, Operating Systems Concepts Essentials

| **Message passing** | **Shared memory** |
|---|---|
| - Użyteczny do wymiany niewielkich ilości danych (brak konieczności zapobiegania konfliktom)<br>- Łatwiejszy w implementacji | - Najszybsza możliwa komunikacja<br>- Jedyna wymagana interwencja z kernela to utworzenie tej pamięci |

Model konsument-klient:

- serwer WWW i przeglądarka HTML
- kompilator tex i pdf viewer

**Shared memory** - bufory wymiany danych:

a) Bufor nieograniczony (*unbounded buffer*) - brak limitu wielkości:
   i) Producent nigdy nie czeka, konsument czeka gdy bufor jest pusty
b) Bufor ograniczony (*bounded buffer*) - określona wielkość bufora:

i) Producent czeka, gdy bufor jest pełny, konsument czeka gdy bufor jest pusty

**Synchronizacja** - *message passing* może być blokujący lub nieblokujący:

a) blokujące wysyłanie
b) nieblokujące wysyłanie
c) blokujący odbiór
d) nieblokujący odbiór

| **Rozmiar bufora** | | |
|---|---|---|
| **Zerowa pojemność** (*zero capacity*) - kolejka ma długość zero | **Ograniczona długość** (*bounded capacity*) - kolejka ma ustaloną długość | **Nieograniczona długość** (*unbounded capacity*) - kolejka ma nieograniczoną długość |

## Wątki

**Wątek** to podstawowa jednostka wykorzystania procesora.

Wątek zawiera:

a) identyfikator (numer)
b) licznik rozkazów
c) rejestry
d) stos.

Wątek współdzieli z innymi wątkami należącymi do tego samego procesu:

a) sekcję kodu
b) sekcję danych
c) inne zasoby systemu operacyjnego (np. otwarte pliki, sygnały).

| **Proces** | **Wątek** |
|---|---|
| - Proces to wykonywanie programu.<br>- *Heavyweight process*. | - Wątek to część danego procesu.<br>- *Lightweight process*. |

| | |
|---|---|
| - Czaso- i zasobochłonne: tworzenie, terminacja, przełączanie kontekstu.<br>- Komunikacja: pamięć dzielona lub wymiana komunikatów jako mechanizmy specjalne.<br>- Procesy są izolowane.<br>- Przełączanie procesów odbywa się przez funkcje systemowe.<br>- Dla jądra dwa procesy to dwa procesy.<br>- Zablokowanie jednego procesu nie wpływa na fakt zablokowania innego procesu.<br>- Zablokowanie się procesu macierzystego uniemożliwia tworzenie procesów potomnych.<br>- Proces ma własny PCB, stos oraz przestrzeń adresową.<br>- Zmiany w procesie macierzystym nie mają wpływu na procesy potomne. | - Szybsze i zużywające mniej zasobów na tworzenie, terminację i przełączanie kontekstu.<br>- Komunikacja: bezpośrednio współdzielone wszystkie zasoby danego procesu.<br>- Wątki współdzielą.<br>- Przełączanie wątków odbywa się bez wywoływania przerwań do jądra.<br>- Dla jądra dwa wątki to jeden proces.<br>- Zablokowanie procesu, to zablokowanie jego wszystkich wątków.<br>- Zablokowanie pierwszego wątku nie wpływa na działanie pozostałych wątków procesu.<br>- Ma rodzica PCB, własny TCB, stos oraz współdzieloną przestrzeń adresową.<br>- Zmiany w procesie wpływają na zmiany w wątkach tego procesu. |

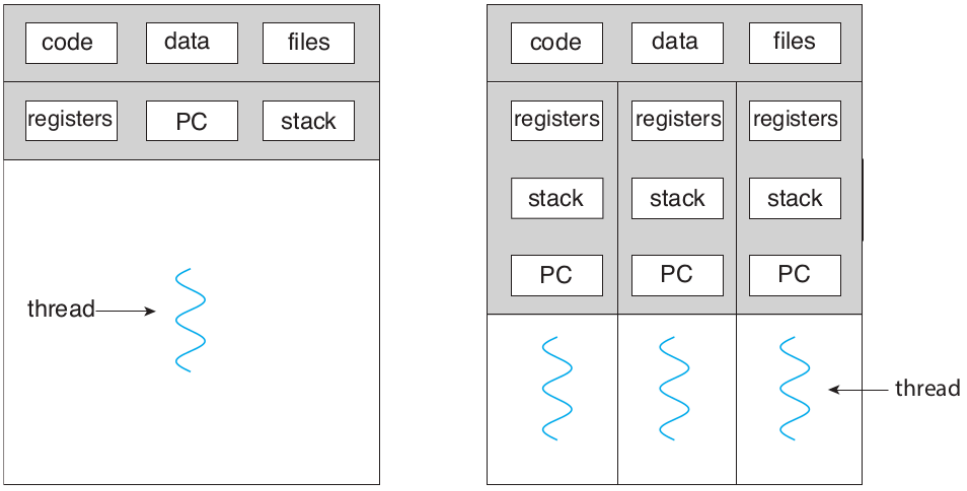
**Obserwowanie wątków w systemie Linux:**

```
$ ps -T -o pid,tid,comm -p pid_procesu
$ pstree -p pid_procesu
```

Dla wątków kernela:

```
$ pstree -p 2
```

*Mają one odmienne TID.*




Źródło: A. Silberschatz, Operating Systems Concepts Essentials

| **PID (*process identifier*)** | **TID (*thread identifier*)** |
|---|---|
| - Jeśli proces ma tylko jeden wątek, to PID == TID<br>- Jeśli proces ma wiele wątków, to pierwszy z nich ma unikalny TID w zakresie tego procesu<br>- Jądro systemu nie rozróżnia szczególnie wątku od procesu<br>- Dla jądra wątki to procesy, które współdzielą pewne zasoby<br>- Kiedy tworzymy nowy proces za pomocą fork(), to otrzymuje on nowy PID i TID (PID == TID)<br>- Kiedy tworzymy nowy wątek to otrzymuje on PID taki jak procesu oraz nowy TID.<br>- Alias: LWP = Light-Weight Process | |

**Wątki na poziomie użytkownika:**
Zarządzanie wątkami odbywa się na poziomie aplikacji. Jądro nie ma wiedzy na temat wątków.

**Wątki na poziomie jądra:**
Jądro zarządza kontekstem dla procesu oraz wątków. Nie ma zarządzania wątkami na poziomie aplikacji.

Zalety:

a) Kernel może jednocześnie planować realizację wielu wątków z jednego procesu na wielu procesorach/rdzeniach.
b) Jeśli jeden wątek w procesie jest zablokowany, jądro może planować inny wątek tego samego procesu.
c) Funkcjonalność jądra może być wielowątkowa.

Wada:

a) przekazanie kontroli między wątkami w obrębie procesu wymaga kernel-mode.

*Istnieje rozwiązanie łączone – tworzenie wątku odbywa się w przestrzeni użytkownika, planowanie (scheduling) oraz synchronizacja wątków odbywa się w jądrze*

**Zastosowanie wątków:**

a) Program serwera obsługujący żądania (requests) programów klienckich:
   i) Serwer stron WWW i przeglądarka internetowa.
   ii) Serwer poczty elektronicznej (mail transfer agent) i program pocztowy.
   iii) Sprawdzanie pisowni w edytorze tekstu.
b) Prowadzenie obliczeń macierzowych:
   i) Wykonywanie operacji na tych samych danych.

**Uwaga! Tworzenie wątku jest mniej obciążające niż tworzenie nowego procesu.**

**Wielowątkowość** – zdolność systemu operacyjnego do wspierania wielu ścieżek wykonywania w obrębie jednego procesu.

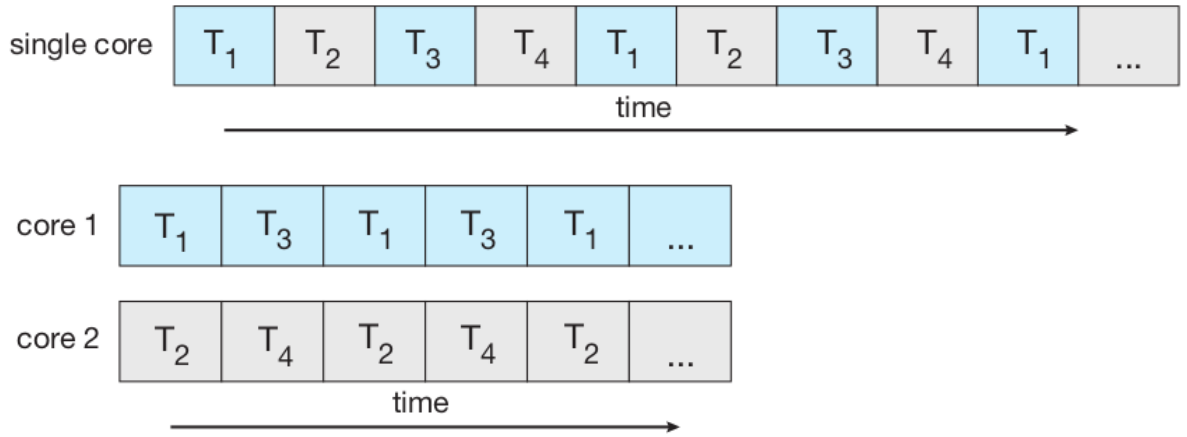
| MS-DOS | Some UNIX | JRE | Windows, Solaris, modern UNIX, Linux, etc. |
|---|---|---|---|
| Jeden proces jednowątkowy | Wiele procesów jednowątkowych | Jeden proces wielowątkowy | Wiele procesów wielowątkowych |

**Zalety wielowątkowości:**

a) **Responsywność** – jeśli część aplikacji jest zablokowana, inna jej część może wykonywać operacje, a cała aplikacja sprawia wrażenie ciągłego działania. Zastosowanie: w aplikacji, kiedy jedna wywołana operacja wykonywana jest w tle, interfejs użytkownika pozostaje responsywny.

b) **Współdzielenie zasobów** - w przypadku procesów współdzielenie zasobów odbywa się tylko poprzez pamięć współdzieloną, albo przesyłanie komunikatów. Wątki współdzielą zasoby wprost. Współdzielenie kodu i danych umożliwia wątkom działać w tej samej przestrzeni adresowej.
c) **Ekonomia** - alokowanie pamięci i zasobów przy tworzeniu procesu jest bardziej kosztowne, niż w przypadku wątków. Przełączanie przełączanie kontekstu jest także szybsze w przypadku wątków.
d) **Skalowalność** - aplikacje wielowątkowe mogą działać w architekturze wielordzeniowej. Aplikacja jednowątkowa może być wykonana tylko na jednym rdzeniu procesora.

| | |
|---|---|
| **Współbieżność** (*concurrency*) | Umożliwia więcej niż jednemu zadaniu być wykonywanym. Do realizacji współbieżności nie jest wymagany system wielordzeniowy |
| **Równoległość** (*parallelism*) | Umożliwia więcej niż jednemu zadaniu być wykonywanym **jednocześnie**. Do realizacji równoległości jest wymagany system wielordzeniowy. |




Źródło: A. Silberschatz, Operating Systems Concepts Essentials

**Wyzwania programowe:**

a) **Identyfikacja zadań,** w szczególności na zadania niezależne między sobą.
b) **Balansowanie** zadaniami celem zrównoważenia obciążenia.
c) **Dzielenie danych** między wydzielone zadania.
d) **Zależność danych** występująca w szczególności przy następstwie obliczeń.
e) **Testowanie i debugowanie** są zdecydowanie trudniejsze niż w programach jednowątkowych.

W trybie **równoległości**:

- **dane** dzielone są między procesami rdzeniami wykonującymi **tego samego typu** operacje
- **zadania** dzielone są między procesami/rdzeniami, a każdy wątek realizuje **unikalną** operację

**Prawo Amdahla**

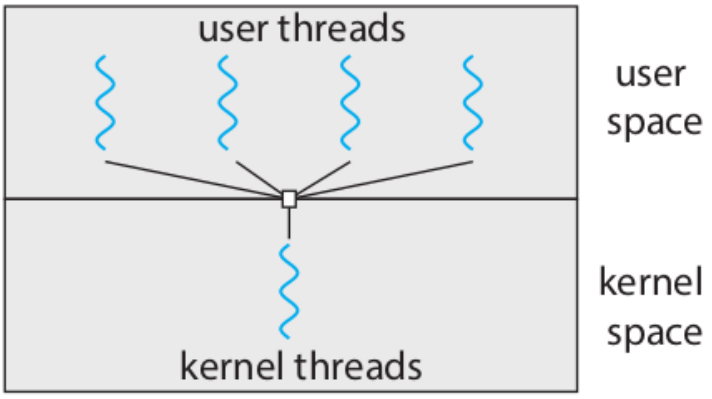
Wyraża potencjalny wzrost wydajności obliczeń przez dodanie kolejnych rdzeni obliczeniowych do obsługi aplikacji, która ma dwa komponenty: podlegający i niepodlegający zrównolegleniu.

**S** - procentowy udział niepodlegającego zrównolegleniu kodu.
**N** - liczba rdzeni przypisanych do zadania.

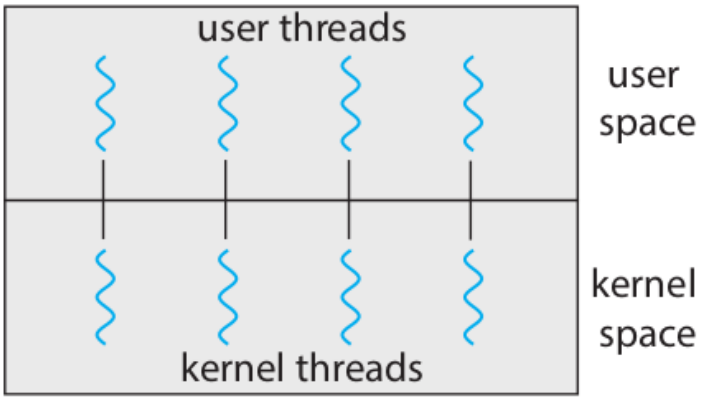
$$speedup \leq \frac{1}{S+\frac{(1-S)}{N}}$$

## Modele wielowątkowe

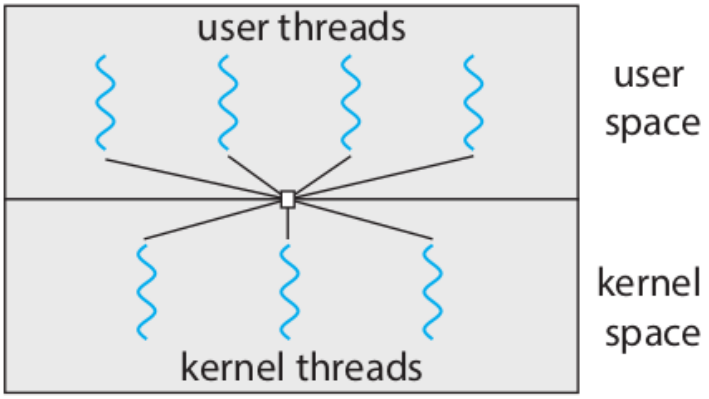
**Many-to-One**



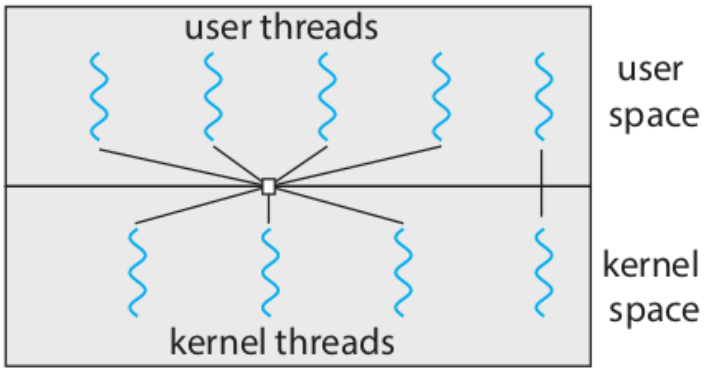

**One-to-One**



**Many-to-Many**




**Two-level**




Źródło: A. Silberschatz, Operating Systems Concepts Essentials

| | |
|---|---|
| **Many-to-One** | Zarządzanie wątkami wykonywane jest w przestrzeni użytkownika (przez bibliotekę).<br><br>Zalety:<br>- wydajność<br>Wady:<br>- zablokowanie całego procesu, jeśli któryś wątek wykona blokujące wywołanie systemowe<br>- wątki nie zostaną uruchomione równolegle w systemie wielordzeniowym, to tzw. zielone wątki (*green threads*) - można je uruchomić w środowisku nie wspierającym wielowątkowości.<br><br>Przykłady: Solaris, wczesne wersje Java. |
| **One-to-One** | Model wprowadza niezależną współbieżność, tzn. dany wątek może być realizowany także wtedy, gdy inny wywoła blokującą funkcję systemową. |

| | |
|---|---|
| | Model wprowadza także równoległość.<br><br>Wady:<br>- każdy wątek użytkownika tworzy wątek w jądrze, a duża ich liczba może obniżać wydajność systemu.<br><br>Przykłady: Linux, Windows. |
| **Many-to-Many** | Model multiplexuje wiele wątków w przestrzeni użytkownika z równą lub mniejszą liczbą wątków w przestrzeni jądra. Liczba wątków w przestrzeni jądra może być specyficzna względem aplikacji lub sprzętu.<br><br>Model ten jest pozbawiony wad modeli Many-to-One i One-to-One.<br><br>Trudny w implementacji. |

**Strategie tworzenia wątków:**

| **Strategia asynchroniczna** – wątek tworzy wątek potomny i następnie kontynuuje swoje działanie. Oba wątki działają współbieżnie i niezależnie. | **Strategia synchroniczna** – wątek tworzy wątki potomne i przechodzi w stan oczekiwania na zakończenie wykonywania ich zadań. O ile wątki potomne działają współbieżnie, to wątek macierzysty po prostu czeka. |
|---|---|
| Zastosowanie:<br>- Serwery wielowątkowe.<br>- Responsywny interfejs użytkownika. | Zastosowanie:<br>- Obliczenia z przesłaniem zadań cząstkowych i oczekiwaniem na wyniki. |

## Pula wątków

Problem:

- Tworzenie wątków zajmuje pewien czas (mniejszy niż procesów potomnych), a może mogą być wykorzystane ponownie.
- Brak kontroli liczby powstających wątków może doprowadzić do przeciążenia zasobów (procesor, pamięć) systemu.

Rozwiązanie: **pula wątków (*thread pool*)**

a) Utworzenie zadanej liczby wątków.

b) Umieszczanie wątków w puli wątków.
c) Wątki oczekują na przydzielenie zadania.
d) Serwer otrzymuje żądanie.
e) Serwer przekazuje żądanie do puli wątków.
f) Jeśli w puli jest wolny wątek, przejmuje on żądanie i zajmuje się jego obsługą.
g) Jeśli brak jest wolnych wątków w puli, zadanie jest kolejkowane.
h) Po zakończeniu obsługi danego żądania wątek wraca do puli i oczekuje na nowe.
i) Pula wątków najlepiej działa, gdy zadania obsługiwane są asynchronicznie.

**Rozmiar puli wątków** może być zależny od:
- Liczby rdzeni procesora.
- Ilości fizycznej pamięci RAM.
- Może być też dynamicznie zmieniany w zależności od aktualnie działających wątków (obserwując ich obciążenie).

**Proces powstały w wyniku wywołania funkcji systemowej fork() jest jednowątkowy!** Wątek wywołujący procesu nadrzędnego staje się initial wątkiem procesu potomnego. Wywołanie exec() spowoduje zastąpienie całego procesu i wszystkich wątków.

## Obsługa sygnałów

**Procedura obsługi** sygnałów:
- Sygnał jest generowany przez zdarzenie.
- Sygnał jest dostarczany do procesu.
- Proces musi obsłużyć sygnał:
  - **domyślna obsługa sygnału** - jeśli brak zdefiniowanej obsługi sygnału, zajmuje się nią jądro systemu operacyjnego (sygnał może być zignorowany lub zakończyć działanie programu),
  - **zdefiniowana przez użytkownika obsługa sygnału.**

Rodzaje sygnałów:

| **Sygnał synchroniczny:** | **Sygnał asynchroniczny:** |
|---|---|
| np. dzielenie przez 0 lub nielegalny dostęp do pamięci | np. wciśnięcie [*Ctrl + c*] |

**Zgodnie z *POSIX*, to który wątek odbierze sygnał nie jest określone.**

Dostarczanie sygnału:

a) Do wątku, do którego sygnał pasuje.
b) Do każdego wątku w procesie.
c) Do wybranych wątków w procesie.
d) Wybrać jeden wątek do przechwytywania wszystkich sygnałów danego procesu.

**Wysłanie sygnału do procesu:**

```
kill(pid_t pid, int signal);
pthread_kill(pthread_t tid, int signal);
```

## Podstawy komunikacji międzyprocesowej

Procesy uruchomione jednocześnie mogą być:

a) **Niezależne** (*independent*) - nie współdzielą danych z żadnym innym wykonywanym procesem w systemie operacyjnym.
b) **Współpracujące** (*cooperating*) - może wpływać lub można na niego wpływać przez inne procesy wykonywane w systemie.

*Proces odczytujący dane z dysku może być procesem współpracującym jak i niezależnym, zależy to od tego czy proces ma dostęp bezpośredni czy pośredni do dysku.*

Zastosowania komunikacji międzyprocesowej:

- Współdzielenie informacji
- Przyspieszenie obliczeń
- Modularność oprogramowania

**Metody dzielenia informacji:**

| Przez system plików | Przez współdzielone informacje z jądra | Przez współdzieloną pamięć |
|---|---|---|

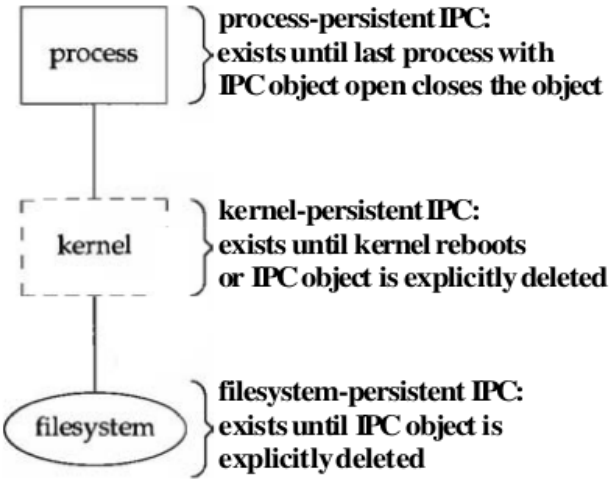
**Implementacja IPC:**

a) Wymiana przez pliki
b) Pamięć dzielona
c) Sygnały
d) Potoki nazwane lub nienazwane
e) Semafory
f) Kolejki
g) Gniazda dziedziny UNIX

h) Gniazda udp/tcp
i) RPC

Trwałość obiektów IPC:

| Type of IPC | Persistence |
|---|---|
| Pipe | process |
| FIFO | process |
| Pcsix mutex | process |
| Posix condition variable | process |
| Posix read–write lock | process |
| `fcntl` record locking | process |
| Posix message queue | kernel |
| Pcsix named semaphore | kernel |
| Pcsix memory-based semaphore | process |
| Posix shared memory | kernel |
| System V message queue | kernel |
| System V semaphore | kernel |
| System V shared memory | kernel |
| TCP socket | process |
| UDP socket | process |
| Unix domain socket | process |




Źródło: R. Stevens, Unix Network Programming - IPC

**Message-Passing Systems (*systemy przekazywania wiadomości*):**

a) Komunikujące się procesy mogą rezydować na różnych stacjach
b) Komunikujące się procesy mogą rezydować na różnych typach i wersjach systemów operacyjnych
c) Infrastruktura systemu przekazywania wiadomości obejmuje co najmniej dwie operacje:
   i) *send*(message)
   ii) *receive*(message)
d) Ze względu na wielkość wiadomości rozróżniamy:
   i) **System bezpośredni**, czyli o stałej długości komunikatów (*straight-forward*) - prosta implementacja w systemie operacyjnym, skomplikowane użytkowanie ze względu na fragmentację,
   ii) **System o zmiennej długości komunikatów** (*variable-sized*) - skomplikowana implementacja w systemie operacyjnym, proste użytkowanie.

**Communication link** - logiczne powiązanie między procesami zestawiające kanał komunikacji między nimi. Nie interesuje nas zatem, czy to jest shared memory, szyna sprzętowa, czy sieć.

Implementacje tego powiązania obejmują zagadnienia:

| **Komunikacja bezpośrednia** | **Komunikacja pośrednia** |
|---|---|
| Każdy proces, który uczestniczy w komunikacji musi bezpośrednio wskazać nadawcę, czy odbiorcę:<br>- send(P, message) - wyślij wiadomość do procesu P.<br>- receive(Q, message) - odbierz wiadomość od procesu Q.<br><br>Własności:<br>- Link jest zestawiany automatycznie, a procesy muszą tylko znać swoją nazwę.<br>- Link jest zestawiany dokładnie między dwoma procesami.<br>- Między każdą parą komunikujących się procesów zestawiany jest dokładnie jeden link.<br>- Występuje symetria w adresacji (P, Q), choć spotykane są warianty asymetryczne (np. P i id). | Procesy komunikują się przez skrzynki (mailbox) lub porty identyfikowane np. przez liczby całkowite:<br>- send(A, message) - wyślij wiadomość do skrzynki A.<br>- receive(A, message) - odbierz wiadomość ze skrzynki A.<br><br>Własności:<br>- Link jest zestawiany między parą współdzielących skrzynkę procesów.<br>- Link jest może zostać zestawiony przez większą liczbę procesów.<br>- Między każdą parą komunikujących się procesów mogą istnieć różne linki komunikacyjne.<br><br>System operacyjny musi obsłużyć następujące mechanizmy:<br>- Utworzenie skrzynki.<br>- Wysyłanie i odbieranie wiadomości do i ze skrzynki.<br>- Usunięcie skrzynki. |
| **Komunikacja synchroniczna (blokująca)** | **Komunikacja asynchroniczna (nieblokująca)** |
| **Blokujące wysyłanie** - proces wysyłający jest zablokowany na wysyłaniu, aż wysyłana wiadomość zostanie odebrana przez proces odbierający lub skrzynkę. | **Nieblokujące wysyłanie** - proces wysyłający wysyła wiadomość i nie jest blokowany na metodzie wysyłającej. |

| **Blokujący odbiór** - odbiorca zostaje zablokowany do czasu otrzymania wiadomości.<br><br>W przypadku blokującego nadawcy i odbiorcy mamy do czynienia z *Rendezvous*. | **Nieblokujący odbiór** - odbiorca otrzymuje gotową wiadomość lub informację o braku jej dostępności. |
|---|---|

| **Buforowanie** | | |
|---|---|---|
| **Pojemność zerowa** | **Ograniczona pojemność** | **Nieograniczona pojemność** |
| Maksymalna długość kolejki wynosi zero, czyli link komunikacyjny nie może mieć żadnej wiadomości oczekującej w sobie, co oznacza, że nadawca musi zostać zablokowany do czasu odbioru wiadomości przez odbiorcę. | Kolejka ma określoną, skończoną długość n, czyli co najwyżej n wiadomości może zostać umieszczonych w kolejce. Jeśli kolejka nie jest pełna, można dołożyć wiadomość (nadawca zostaje zablokowany), jeśli kolejka jest pusta, nie można pobrać żadnej wiadomości (odbiorca zostaje zablokowany). | Długość kolejki jest potencjalnie nieskończona (nadawca nigdy nie jest blokowany). |

## POSIX Shared Memory

| | |
|---|---|
| **Tworzenie dowiązania do wspólnej przestrzeni w pamięci** | Pisanie i czytanie:<br>*int fd = shm_open(name, O_CREAT \| O_RDWR, 0666);*<br><br>Tylko czytanie:<br>*int fd = shm_open(name, O_RDONLY, 0666);* |
| **Ustawianie wielkości pamięci dzielonej** | *ftruncate(fd, 4096);* |

| | |
|---|---|
| **Utworzenie miejsca w pamięci** | *ptr = (char *) mmap (0, SIZE, PROT_READ \| PROT_WRITE, MAP_SHARED, fd, 0);* |
| **Pisanie do pamięci dzielonej** | Umieszczenie komunikatu w miejscu 'ptr':<br>*sprintf(ptr, "%s", message);*<br><br>Przesunięcie wskazania w pamięci:<br>*sprintf(ptr, "%s", message);* |
| **Czytanie z pamięci dzielonej** | *printf("%s", (char *)ptr);* |
| **Usunięcie obiektu współdzielonego** | *shm_unlink(name);* |

| Description | mq_open | sem_open | shm_open |
|---|---|---|---|
| read-only<br>write-only<br>read-write | O_RDONLY<br>O_WRONLY<br>O_RDWR | | O_RDONLY<br><br>O_RDWR |
| create if it does not already exist<br>exclusive create | O_CREAT<br>O_EXCL | O_CREAT<br>O_EXCL | O_CREAT<br>O_EXCL |
| nonblocking mode<br>truncate if it already exists | O_NONBLOCK | | <br>O_TRUNC |

| | Message queues | Semaphores | Shared memory |
|---|---|---|---|
| Header | <sys/msg.h> | <sys/sem.h> | <sys/shm.h> |
| Function to create or open | msgget | semget | shmget |
| Function for control operations | msgctl | semctl | shmctl |
| Functions for IPC operations | msgsnd<br>msgrcv | semop | shmat<br>shmdt |

Źródło: R. Stevens, Unix Network Programming - IPC

## Potok (pipe)

Jedne z pierwszych metod IPC, jedna z najprostszych form komunikacji.

Przy projektowaniu potoków należy uwzględnić cztery kwestie:

1) Czy potok pozwala na dwukierunkową (*bidirectional*), czy jest to jednokierunkowa (*unidirectional*) komunikacja?
2) Jeśli możliwa jest komunikacja dwukierunkowa, to czy jest ona **half duplex** (dane przesyłane są tylko w jednym kierunku w danym momencie), czy **full duplex** (dane przesyłane są w obu kierunkach w danym momencie)?
3) Czy jest relacja między komunikującymi się procesami (np. rodzic - dziecko)?
4) Czy potoki mogą komunikować się poprzez sieć, czy tylko między procesami na tej samej maszynie?

| | |
|---|---|
| **Utworzenie potoku** | *int fd[2];*<br>*pipe(int fd[]);* |
| **Pisanie do potoku (deskryptor fd[1])** | *write(fd[WRITE END], write msg, strlen(write msg)+1);* |
| **Czytanie z potoku (deskryptor fd[0])** | *read(fd[READ END], read msg, BUFFER SIZE);* |

**W praktyce:**

```
$ ls | less
$ cat file.txt | wc -l
```

*Ponieważ fork() dziedziczy od procesu rodzica deskryptory, a potok jest deskryptorem, zaleca się odpiąć potok od rodzica, jeżeli dziecko ma go obsługiwać.*

## Kooperacja procesów

| | |
|---|---|
| **Sposoby kooperacji:** | - Bezpośrednie współdzielenie przestrzeni adresacji (zarówno kod, jak i dane)<br>- Współdzielenie danych przez system plików lub komunikaty |
| **Skutki kooperacji:** | - Utrata spójności danych<br>- Wzajemne blokowanie |

**Sekcja krytyczna** (*critical section*) to segment kodu, w którym proces może zmieniać wartości zmiennych, aktualizować tabele, pisać do pliku, etc. Podstawową własnością sekcji krytycznej jest to, że w tym samym czasie żaden inny proces nie może realizować swojej sekcji krytycznej (obejmującej te same zasoby).

a) **Sekcja wejścia** (*entry section*) - segment kodu, w którym zgłaszane jest żądanie dostępu do zasobu celem realizacji wzajemnego wykluczenia.
b) **Sekcja krytyczna** (*critical section*)
c) **Sekcja wyjścia** (*exit section*) - segment kodu, w którym zgłaszane jest zwolnienie zasobu.
d) **Sekcja pozostałego kodu** (*remainder section*) - nie związana z obsługą współdzielenia zasobów część pozostała część kodu.

Rozwiązanie problemu sekcji krytycznej musi spełniać następujące wymagania:

- **Wzajemne wykluczenie** (*mutual exclusion*) oznacza, że jeśli jeden proces wykonuje swoją sekcję krytyczną, to żaden inny proces nie może wykonać swojej sekcji krytycznej.
- **Postęp** (*progress*) oznacza, że jeśli żaden proces nie jest w sekcji krytycznej i jakiś proces chciałby wejść do swojej sekcji krytycznej, to tylko procesy nierealizujące swojej sekcji kodu pozostałego mogą brać udział w decydowaniu, który z nich wejdzie do swojej sekcji krytycznej.
- **Skończony czas oczekiwania** (*bounded waiting*) oznacza, że czas oczekiwania na wejście do sekcji krytycznej dla każdego procesu powinien być ograniczo

**Zakleszczenie (*deadlock*) - sytuacja, kiedy dwa procesy wzajemnie czekają na zwolnienie zasobów**

**Wywłaszczenie** - technika, w której planista (algorytm szeregujący zadania, *dispatcher*) może wstrzymać aktualnie wykonywane zadanie, aby umożliwić wykonywanie innemu zadaniu. Zawieszenie (np. zapętlenie) zadania nie powoduje zawieszenia całego systemu.

**Wywłaszczenie jądra** - jądro pozwala na wywłaszczenie własnego kodu, co oznacza, że w wykonywanie jego kodu może zostać przerwane na czas wykonywania przez procesor innego zadania.

**Wywłaszczanie przerwań** - przerwania są zwykle niewywłaszczalne, dlatego powinny być krótkie.

| Cecha | Jądro wywłaszczające | Jądro niewywłaszczające |
|---|---|---|
| Definicja | Pozwala na usuwanie i podmianę procesu wykonywanego w trybie kernela, a w efekcie wykonywane jest zadanie o najwyższym priorytecie. | Pozwala na wywłaszczenie procesu wykonywanego w trybie kernela, co oznacza konieczność czekania na jego zakończenie. |
| Warunek wyścigu | Występuje - wiele procesów jest aktywnych | Nie występuje - jeden proces jest aktywny |
| Responsywność | Większa i deterministyczna responsywność | Mniejsza i niedeterministyczna responsywność |
| Implementacja | Skomplikowany projekt i implementacja | Mniej skomplikowany projekt i implementacja |
| Bezpieczeństwo | Większa stabilność pracy i użyteczność | Mniejsza stabilność pracy i użyteczność |
| Semafory | Nie wymaga użycia semaforów | Dane współdzielone wymagają semaforów |
| Programowanie RT | Większa użyteczność w programowaniu RT | Mniejsza użyteczność w programowaniu RT |
| Wywłaszczanie | Jest | Brak |
| Przykłady | Linux od 2.6, IRIX, Solaris, NetBSD od v5<br>Microkernel: Windows NT, Vista, 7 i 10 | Windows XP, Windows 2000, Linux do 2.4 |

**Algorytm Petersona**

Dwa procesy P0 i P1 współdzielą:

- int turn;
- boolean flag[2];

Zmienna turn wskazuje, którego procesu jest kolej na wejście do sekcji krytycznej. Tablica flag wskazuje, czy proces jest gotowy na wejście do sekcji krytycznej.

## Synchronizacja sprzętowa

| **Test and set lock - TSL** |
|---|
| - Wymagane wsparcie procesora do realizacji instrukcji atomowych<br>- Realizacja sekwencyjna instrukcji atomowych (także w przypadku SMP)<br>- Instrukcja atomowa: *TestAndSet()* |
| **Swap** |
| - Wymagane wsparcie procesora do realizacji instrukcji atomowych<br>- Realizacja sekwencyjna instrukcji atomowych (także w przypadku SMP)<br>- Instrukcja atomowa: *Swap()*<br>- Inicjalizacja globalnych zmiennych:<br>  - *boolean waiting[n];*<br>  - *boolean lock;* |
| **TSL + czas oczekiwania** |
| - Połączenie Test and set lock oraz Swap<br>- Spełniają wymaganie wzajemnego wykluczenia, ale nie spełniają wymagania dot. skończonego czasu oczekiwania |
| **Semafory** |
| - Semafor S to zmienna całkowita<br>- Semafor można modyfikować tylko w:<br>  - *wait()*<br>  - *signal()*<br>- Kiedy jeden proces modyfikuje semafor, żaden inny nie może tego robić<br>- Testowanie warunku *S <= 0* oraz inkrementacja *S--* musi wykonać się bez przerwania<br>- Typy semaforów:<br>  - **Semafor binarny** (*binary semaphore*) = *mutex lock* od: *mutual exclusion*, czyli wzajemne wykluczenie Przy |

zastosowaniu do sekcji krytycznej:
  - procesy współdzielą semafor, *mutex = 1*,
  - każdy proces działa jak na listingu obok.
- **Semafor zliczający** (*counting semaphore*)
  Zastosowanie do sekcji krytycznej, kiedy dany zasób ma wiele instancji.

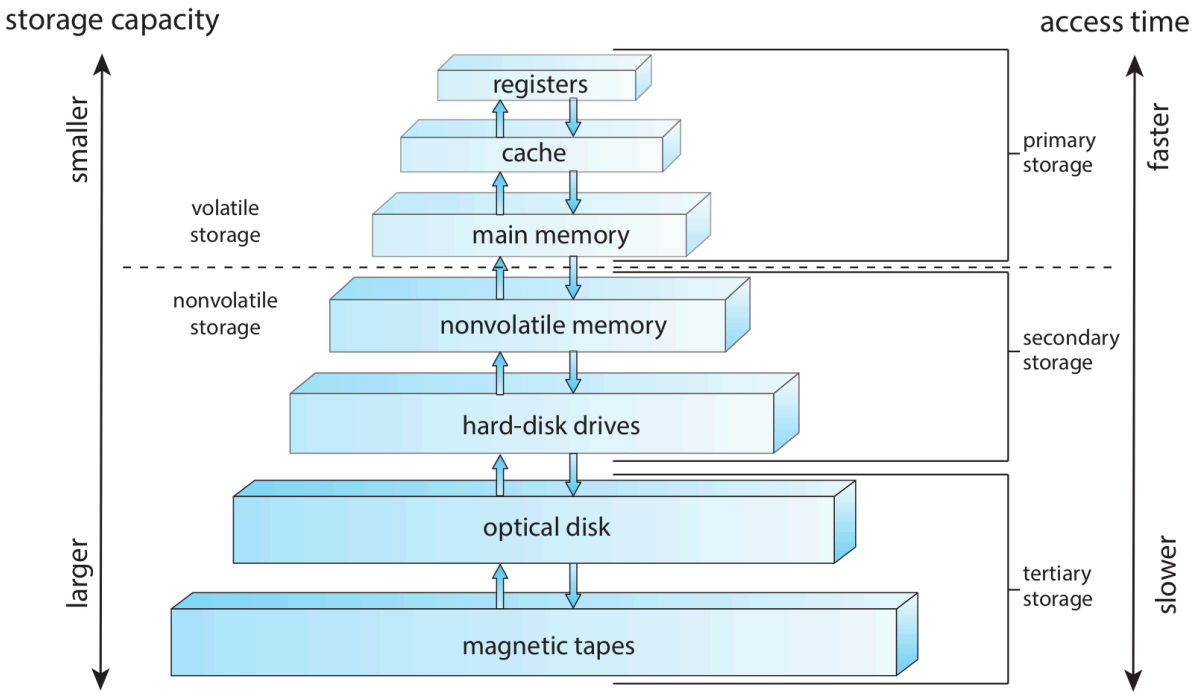
## Problemy synchronizacyjne:

| | |
|---|---|
| **Problem ograniczonego bufora** | Założenia:<br>- Pula buforów, rozmiar puli wynosi *n*<br>- Każdy bufor może zawierać jeden obiekt<br>- Zmienna *mutex* jest semaforem b. do puli<br>- Na początku *mutex = 1*<br>- Semafory *empty* i *full* to liczność pustych/pełnych buforów<br>- Na początku *empty = n*, *full = 0* |
| **Problem czytelników i pisarzy** | Założenia:<br>- Istnieje współdzielona baza danych<br>- Dwa typy procesów: *piszące* i *czytające*<br>- Procesy czytające mogą w dowolnej liczbie osiągać dostęp do bazy<br>- Jeśli jeden proces piszący ma dostęp, w tym czasie żaden inny proces (ani piszący, ani czytający) nie może mieć dostępu<br><br>Warianty:<br>I. Żaden proces czytający nie czeka na dostęp, chyba, że proces piszący go uzyskał. Ryzyko zagłodzenia pisarzy.<br>II. Jeśli pisarz oczekuje na dostęp, żaden czytelnik nie może rozpocząć czytania. Może dojść do zagłodzenia czytelników.<br><br>Rozwiązanie:<br>- Czytelnicy współdzielą:<br>  - *semaphore mutex, wrt;*<br>  *// init: 1*<br>  - *int readcount;* |

| | |
|---|---|
| | *// init: 0*<br>- *wrt* jest wspólny także dla pisarzy<br>- *wrt* jest muteksem obsługującym pisarzy<br>- *wrt* jest także dla pierwszego i ostatniego czytelnika w sekcji krytycznej,<br>- *mutex* obsługuje zmienną *readcount*<br>- *readcount* - liczba aktualnie czytających |
| **Problem ucztujących filozofów** | Założenia:<br>- Rozważmy pięciu filozofów siedzących przy stole jak na obrazku obok<br>- Na środku stołu jest miska ryżu, wokół pięć talerzy, po jednym dla każdego filozofa<br>- Pomiędzy talerzami jest pięć sztućców<br>- Od czasu do czasu dany filozof chce zjeść<br>- Aby zjeść muszą być wolne dwa sztućce<br>- Jedząc filozof ma wyłączność na 2 sztućce<br>- Po skończeniu jedzenia zwalnia sztućce<br><br>Rozwiązania:<br>- Filozof próbuje wziąć szczućce wywołując: *wait()*<br>- Filozof odkłada sztućce wywołując: *signal()*<br>- Filozofowie współdzielą sztućce:<br>- *semaphore chopstick[5];*<br>*// init: 1*<br>- Max 4-ch filozofów przy stole<br>- Można podnieść sztućce tylko wtedy, jeśli oba są wolne (podnieść w sekcji krytycznej)<br>- Parzyści filozofowie podnoszą najpierw lewy, potem prawy sztuciec, a nieparzyści odwrotnie: najpierw prawy, potem lewy |

# Współdzielenie zasobów

| CPU | MEM | HDD | NET |
|---|---|---|---|
| współdzielenie w czasie | współdzielenie w ilości | współdzielenie w dostępie | współdzielenie w czasie |




Źródło: A. Silberschatz, Operating Systems Concepts Essentials

# Procesor + pamięć

Procesor ładuje instrukcje tylko z **pamięci głównej**, więc każdy program musi być do niej najpierw załadowany.

**Pamięć główna** (*main memory*, RAM - *random-access memory*) wykonana jest w technologii półprzewodnikowej zwanej DRAM

Nie wszystko mieści się w pamięci RAM oraz pamięć ta jest ulotna, stąd wymagana jest pamięć dodatkowa, tj. *secondary storage* (*hard-disk drives* - **HDDs** lub *nonvolatile memory* - **NVM**).

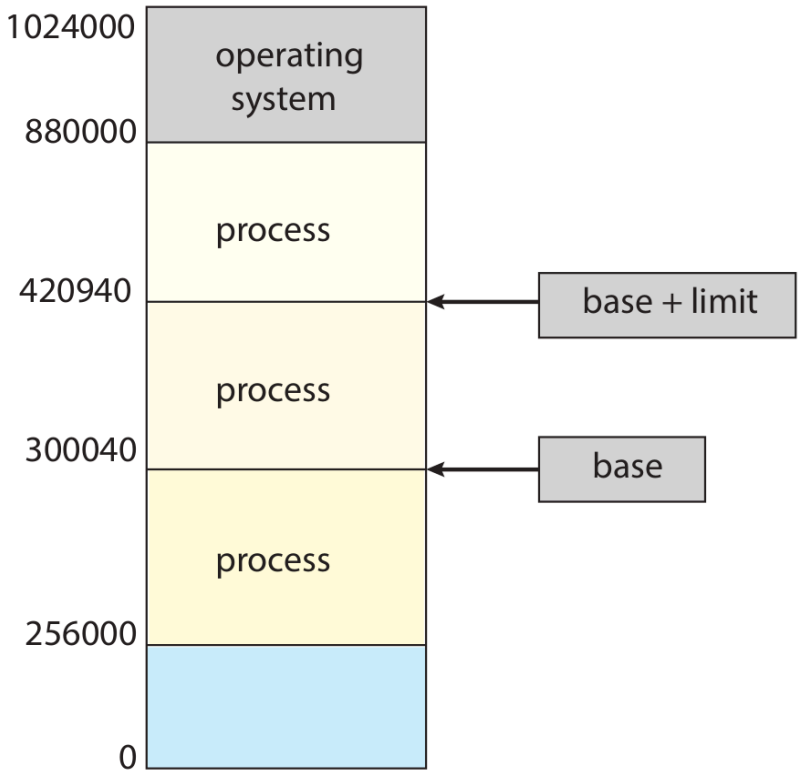
**Pamięć główna (RAM/ROM)** oraz **rejestry wbudowane** w CPU to jedyna pamięć, którą CPU może adresować bezpośrednio.

Jeśli dana, na której mają być wykonywane operacje znajduje się na którejkolwiek z pozostałych pamięci, musi zostać najpierw skopiowana w obszar o bezpośrednim dostępie.

Adresowanie rejestrów, w odróżnieniu od adresowania pamięci głównej, zazwyczaj odbywa się w **jednym takcie procesora**.

Aby w trakcie uzyskiwania dostępu do pamięci głównej procesor nie marnował cykli, wykorzystywany jest *cache*.

## Izolowanie pamięci




Źródło: A. Silberschatz, Operating Systems Concepts Essentials

Izolacja pamięci dla każdego procesu gwarantuje indywidualną przestrzeń.

Pamięć przydzieloną procesowi (na zmienne statyczne) wyznaczają dwa rejestry: **baza** i **limit**.

Proces użytkownika (*user mode*) może zapisywać i odczytywać tylko pamięć w przydzielonym zakresie.

System operacyjny (*kernel mode*) może swobodnie operować po całej pamięci, w szczególności zmieniać bazę i limit. Pozwala to na kontrolowanie innego oprogramowania.

## Wiązanie adresu

Adres w kodzie programu **może być symboliczny**, np. nazwa zmiennej. Kompilator wiąże w/w adres symboliczny do **adresu relokowalnego** (względem danego modułu). Linker lub loader zamienia adres relokowalny w **bezwzględny**.

Wiązanie może wystąpić na każdym z etapów:
- Kompilacja (stare systemy),
- Ładowanie (MS-DOS),
- Wykonywanie - najczęściej (obecnie).

**Rodzaje adresów:**

| | |
|---|---|
| **Adres logiczny** - adres widziany przez proces i dla niego dostępny. | **Adres fizyczny** - adres na szynie adresowej. |

Translacją adresów zajmuje się **MMU** - *Memory management unit*.

**Wiązanie adresów w czasie:**

| | |
|---|---|
| **kompilacja** | adres logiczny == adres fizyczny |
| **ładowanie** | adres logiczny == adres fizyczny |
| **wykonywanie** | adres logiczny != adres fizyczny |

Adres wirtualny jest równoważny adresowi logicznemu w sytuacji wiązania w czasie wykonywania.

## Linkowanie

| Dynamiczne | Statyczne |
|---|---|
| Biblioteka systemowa linkowana do programu w momencie jego uruchomienia (wymaga wsparcia ze strony | Włączanie biblioteki systemowej do obrazu binarnego programu. |

| | |
|---|---|
| systemu operacyjnego) (**DLL** - *Dynamic-Link Library*).<br><br>Zalety DLL:<br>- nie trzeba kopiować/implementować istniejących już modułów programu (oszczędność pamięci),<br>- biblioteka może zostać jednokrotnie załadowana do pamięci, a wiele procesów może z niej korzystać,<br>- biblioteki mogą być aktualizowane jednokrotnie, jakkolwiek każdy program może używać wskazanej wersji. | |

## Ochrona pamięci

Każdy proces wskazany przez scheduler CPU do uruchomienia sprawdzany jest względem rejestrów:

a) **Relocation register** (rejestr bazowy, rejestr relokacji) - początkowy adres fizyczny obszaru przeznaczonego dla procesu.
b) **Limit register** - zawiera zakres adresów logicznych.

Dzięki temu można chronić system operacyjny i inne programy przed modyfikacją przez ten program.

## Alokacja pamięci

Metoda **variable-partition** - przypisanie procesu do ciągłego obszaru pamięci pamięci i pamiętanie przez SO, które części pamięci są zajęte, a które wolne.

Jeśli nie ma możliwości umieścić proces w pamięci:
- proces nie jest uruchamiany
- proces oczekuje w kolejce.

Zarządzanie wolną przestrzenią (*hole*) obejmuje jej dzielenie i łączenie i nazywane jest *dynamic storage allocation problem*:

a) **First fit** - pierwszy pasujący
b) **Best fit** - najlepiej pasujący
c) **Worst fit** - największy wolny

**Fragmentacja:**
W przypadku strategii *first-fit* i *best-fit* szybko dochodzi do tzw. fragmentacji zewnętrznej.

Najgorszy przypadek fragmentacji: między każdą dwójką procesów jest wolna przestrzeń.

Oprócz doboru strategii znaczenie ma też położenie nowego procesu (*na początku czy na końcu wolnego miejsca?*).

Statystycznie, po pewnym czasie strategii *first-fit* aż **50%** bloków zmarnowanych będzie na fragmentację.

W przypadku podzielenia pamięci na bloki przydzielona pamięć dla danego procesu może być większa niż proces wymaga - różnica między wielkością przydzieloną a wymaganą to fragmentacja wewnętrzna.

Rozwiązanie:

- **kompaktowanie** (np. Java: garbage collection) - możliwe tylko w czasie działania.
- **stronicowanie**.

**Stronicowanie:**
Stronicowanie powoduje, że logiczna przestrzeń adresowa jest nieciągła.

Pamięć fizyczna dzielona jest na **bloki**, *frames*. Pamięć logiczna dzielona jest na **strony** tej samej wielkości, *pages*.

Kiedy proces ma być wykonywany, jego strony ładowane są do ramek dostępnej pamięci. Każdy proces ma swoją tablicę stron.

Rozmiar strony zależny jest od sprzętu i wynosi od 4KB do 1GB i ze względu na adresację jest potęgą 2 (`$ getconf PAGESIZE`).

# Pamięć wirtualna

Procesy widzą pamięć fizyczną przez pamięć wirtualną.

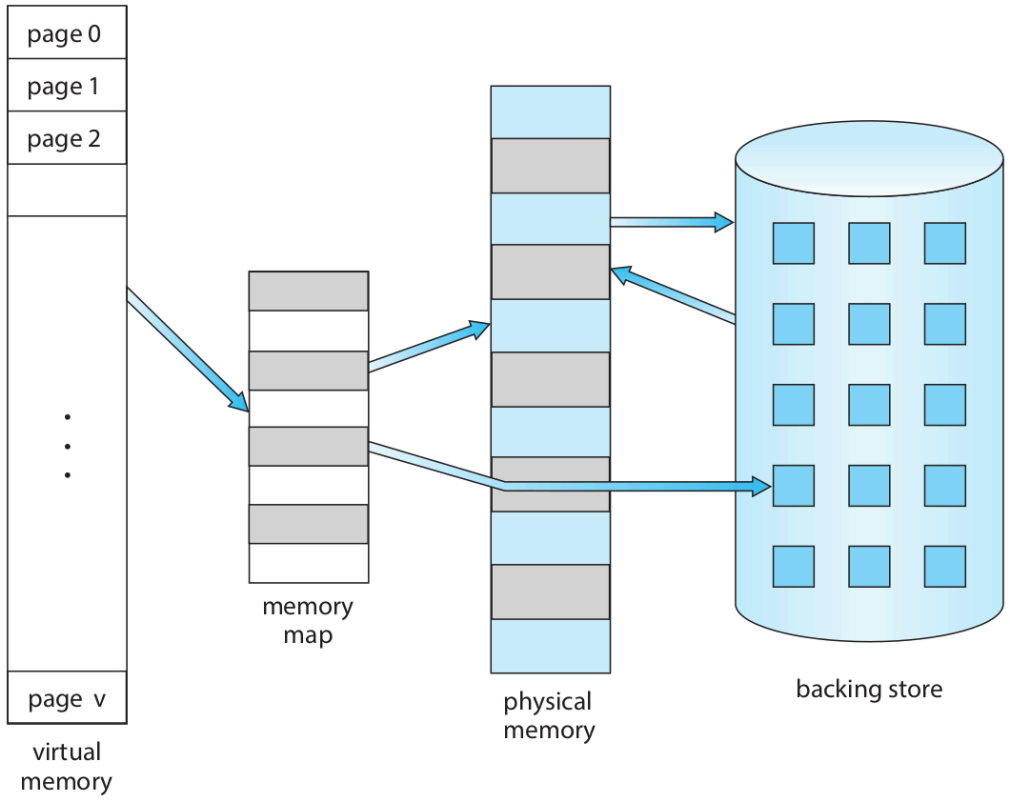
**Pamięć wirtualna** to technika pozwalająca na wykonywanie procesów, które nie są całkowicie w pamięci (m.in. np. z powodu ich rozmiaru).

Pamięć wirtualna jest abstraktem pamięci głównej jako bardzo dużej macierzy, oddzielając pamięć logiczną przeznaczoną dla programisty od pamięci fizycznej.

*Pamięć wirtualna pozwala procesom na* ***współdzielenie plików, bibliotek*** *oraz* ***implementację pamięci dzielonej.***

Pamięć wirtualna oddzielona jest od pamięci fizycznej.Dzięki temu możliwe jest stosowanie olbrzymiej pamięci wirtualnej przy niewielkiej pamięci fizycznej.

Pamięć zapasowa (*backing store*, *spaw space*) - przestrzeń poza pamięcią główną.




Źródło: A. Silberschatz, Operating Systems Concepts Essentials

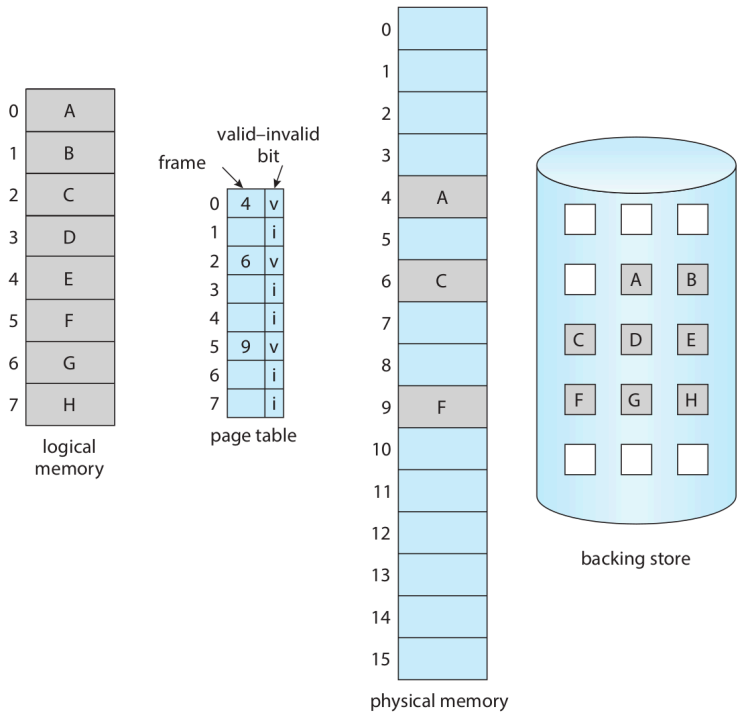
**Wirtualna przestrzeń adresowa**

| Stack | Heap | Data |
|---|---|---|
| Miejsce, w którym dane umieszczane są w sposób uporządkowany, w tym miejscu odkładane są dane dotyczące wywołań funkcji, zmiennych (statyczne w tym globalne, automatyczne w tym lokalne). Miejscem tym zarządza program. | Miejsce, w którym dane umieszczane są swobodnie wg zapotrzebowania, zwykle przez wywołanie malloc/new, są to zmienne (dynamiczne). Miejscem tym zarządza programista. | Obszar o znanym w czasie kompilacji rozmiarze, w którym umieszczane są dane statyczne, w tym globalne. |

Adresy wirtualne bibliotek systemowych mogą być mapowane z różnych procesów (w trybie tylko do odczytu) na wspólną część w przestrzeni adresów fizycznych.

Procesy mogą współdzielić obszar pamięci (*shared memory*) celem wymiany komunikatów i ją też muszą adresować w przestrzeni adresów wirtualnych.

**Stronicowanie na żądanie** – strona jest wprowadzana do pamięci wtedy, gdy jest potrzebna.



Tablica stron zawiera dodatkowo bity poprawności odwołania:
  a) **1** – strona znajduje się w pamięci głównej (*valid*)
  b) **0** – strona znajduje się poza pamięcią główną (*invalid*)

Odwołanie do strony z bitem równym 0:
  - Błąd braku strony (*page fault*)
  - Obsługa błędu braku strony

**Obsługa błędu braku strony**

Gdy referencja ma stan invalid to:
  a) jeżeli strona w ogóle nie istnieje:
      - terminowanie procesu
  b) jeżeli strona jest dostępna w pamięci zapasowej:
      - wysłanie zgłoszenia do systemu operacyjnego o wczytanie strony z pamięci zapasowej
      - odnalezienie na liście wolnej ramki (system operacyjny zwykle stosuje technikę *zero-fill-on-demand*, która zeruje ramki przed użyciem; w momencie startu systemu, cała dostępna pamięć umieszczana jest na liście wolnych ramek) i wczytanie strony do tej ramki
      - aktualizacja tablicy stron
      - restart instrukcji, która wywołała błąd

**Przebieg stronicowania na żądanie:**

1. Odwołanie/przerwanie do systemu operacyjnego.
2. Zapisz stan rejestrów oraz procesu.
3. Stwierdź, że przerwanie wywołane było przez błąd strony.
4. Sprawdź, czy odwołanie jest właściwe i określ położenie strony w pamięci zapasowej.
5. Wywołaj odczyt z pamięci zapasowej do wolnej ramki:
    a. Czekaj w kolejce, aż żądanie odczytu zostanie obsłużone.
    b. Odczekaj czas działania urządzenia.
    c. Rozpocznij transfer strony do wolnej ramki.
6. Podczas oczekiwania, przekaż CPU innemu procesowi.
7. Przechwyć przerwanie z podsystemu I/O (zakończenie wczytywania).
8. Zapisz rejestry oraz stan innego procesu (jeśli krok 6 jest wykonany).
9. Określ, że przerwanie było z pamięci zapasowej.
10. Zaktualizuj tablicę stron, aby wskazać, że określona strona jest teraz w pamięci.
11. Czekaj, aż CPU zostanie przypisany znów temu procesowi.

12. Wznów rejestry, stan procesu oraz nową tablicę stron i wznów działanie procesu.

***Problem: Stronicowanie na żądanie zajmuje bardzo dużo czasu!***

**Copy-on-Write** w technice *fork()* pozwala na współdzielenie tych samych stron w pamięci. Tylko ta strona, która jest modyfikowana wymaga kopiowania (dopiero po jej modyfikacji przydzielane jest jej nowe miejsce w pamięci). Często po *fork()* występuje *exec()* i wtedy okazuje się, że kopiowanie w ogóle nie jest potrzebne.

**Over-allocation** to termin, który odnosi się do sytuacji, w których zasoby są alokowane na nadmiernym poziomie.

## Zastępowanie stron

1. Znajdź stronę w pamięci zapasowej.
2. Szukaj wolnej ramki:
    a. Jeśli jest, użyj jej.
    b. Jeśli brak, użyj algorytmu wyboru ramki podlegającej wymiane (victim frame).
    c. Zapisz jej zawartość do pamięci zapasowej, zaktualizuj tablice ramek i stron.
3. Wczytaj oczekiwaną stronę do zwolnionej ramki. Zaktualizuj tablice ramek i stron.
4. Kontynuuj działanie procesu.

**Celem optymalizacji stosuje się bit modyfikacji, tzn. brudny bit (*dirty bit*). Np. strony z kodem programu zasadniczo są read-only.**

**Algorytmy zastępowania stron:**

| | |
|---|---|
| **Algorytm FIFO** | Najstarsza z umieszczonych w pamięci stron (głowa) jest zastępowana, nowe strony umieszczane są na końcu kolejki (ogon),<br><br>Wady:<br>- z jednej strony strona zastępowana może być czymś, co było dawno temu umieszczone w pamięci i jest już nieużywane, ale może to być też często używana zmienna istniejaca od początku procesu. |

| | |
|---|---|
| **Optymalne zastępowanie stron** | Zastąp stronę, która nie będzie używana przez najdłuższy czas.<br><br>Wady:<br>- trzeba znać przyszłość |
| **Algorytm LRU - *least recently used*** | Zastąp stronę, która najdłużej nie była używana,<br><br>Implementacje:<br>- liczniki (timestamp ostatniego użycia)<br>- stos (przekładanie na wierzch użytej strony) |

**Strategie alokacji ramek:**

a) **Minimalna liczba ramek** - określona jest minimalna liczba ramek, które muszą zostać zaalokowane. Powód: im mniej zaalokowanych ramek, tym bardziej spada wydajność wykonywanych procesów.

b) **Równa alokacja** - ramki po równo podzielone są między procesy

c) **Proporcjonalna alokacja** - ramki przydzielane są proporcjonalnie do wielkości procesów.

| **Alokacja globalna / zastępowanie globalne** | **Alokacja lokalna / zastępowanie lokalne** |
|---|---|
| Realizacja procesu wymaga zastępowania ramek pośród wszystkich ramek, także tych zaalokowanych do innego procesu.<br><br>Alokacja globalna ma szczególne zastosowanie w przypadku priorytetyzowania procesów. | Realizacja procesu wymaga zastępowania ramek tylko pośród zaalokowanych do procesu. |

**Major page fault** - występuje wtedy, gdy strona jest wywoływana, a nie znajduje się w pamięci. Obsługa tego błędu wymaga odczytania wskazanej strony z pamięci zapasowej do wolnej ramki i aktualizacji tablicy stron. Stronicowanie na żądanie zwykle generuje znaczną liczbę tych błędów.

**Minor page fault** - występuje wtedy, gdy proces nie ma logicznego mapowania do strony, ale ta strona jest w pamięci. Błąd ten występuje w jednym z przypadków:

- Proces odwołuje się do biblioteki dzielonej, która jest w pamięci, ale proces nie ma do niej mapowania. W tym przypadku wystarczy zaktualizować tablicę stron.
- Proces utracił stronę, która trafiła na listę wolnych ramek, ale nie została jeszcze wyzerowana. W tym przypadku strona jest ponownie przypisana do procesu i usunięta z listy wolnych ramek.

Przydatne polecenie:

```
$ ps -eo min_flt, maj_flt, cmd
```

**Przydatne komendy:**

ls, du, stat, mkfs.[ext4/fat]

Lokalizacja plików urządzenia pamięci:

- $ cat /proc/meminfo,
- $ ls -alh /proc/kcore

Sprawdzić położenie pamięci swap:

- cat /etc/fstab | grep swap
- sudo fdisk -l /dev/sda

Zawartość /proc/[pid]/

- /proc/[pid]/cmdline Nazwa polecenia za pomocą którego utworzono proces
- /proc/[pid]/io Statystyki operacji wejścia wyjścia
- /proc/[pid]/status Status procesu, wykorzystanie pamięci
- /proc/[pid]/limits Limity zasobów
- /proc/[pid]/sched Informacje dotyczące szeregowania procesu
- /proc/[pid]/fdinfo/ Informacje o plikach które proces otworzył
- /proc/[pid]/fd/ Deskryptory plików które proces otworzył
- /proc/[pid]/smaps/
- /proc/[pid]/net/ Informacje dotyczące sieci

Polecenia: ltrace i strace, vmstat

Polecenie pagemon, np.: sudo pagemon -p `pidof xclock` (dla procesu xclock)

Polecenie smem, np.: sudo smem --piename -c “pss”

Tworzenie pliku/partycji wymiany: dd/fdisk, mkswap, swapon

Czyszczenie pamięci:

- # sync; echo 1 > /proc/sys/vm/drop_caches
- 1, 2 lub 3

Oraz swap: swapoff -a && swapon -a