



# Systemy Operacyjne

Komunikacja międzyprocesowa

Dr hab. inż. Krzysztof Rzecki, prof. AGH

Na podstawie: Abraham Silberschatz, *Konceptcje systemów operacyjnych*

Na podstawie: Richard Stevens, *Unix Network Programming - IPC*



# Podstawy komunikacji międzyprocesowej

Procesy uruchomione jednocześnie mogą być:

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

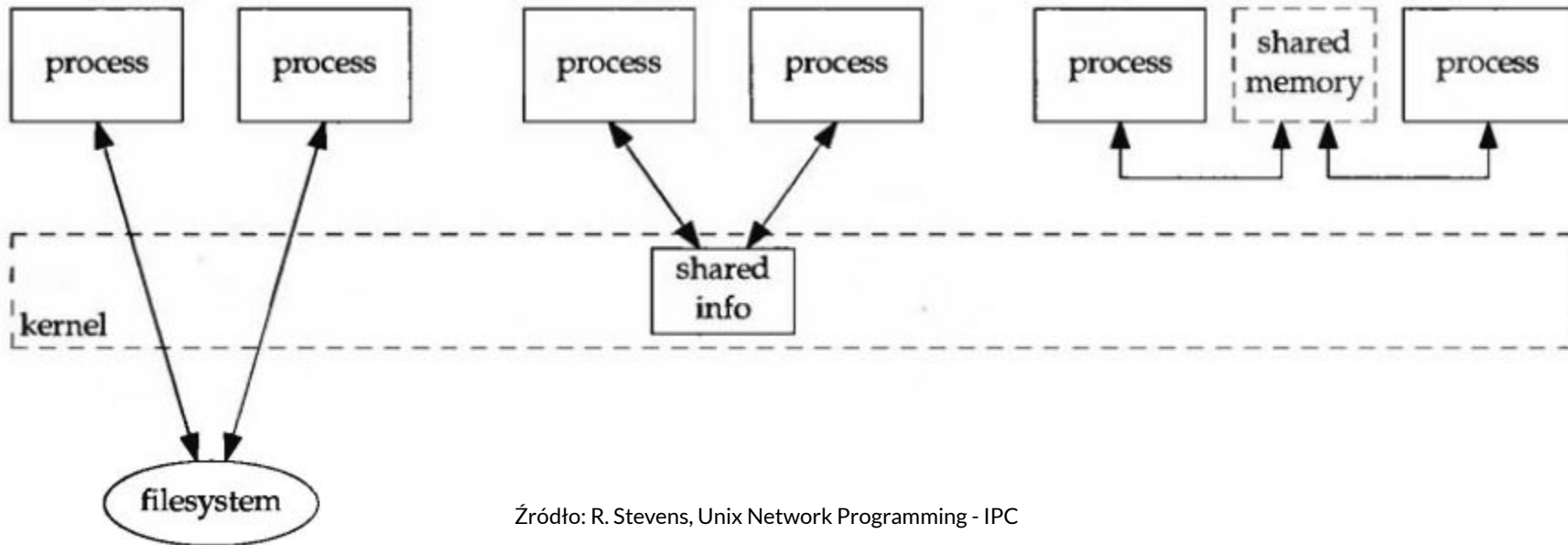
Czy zatem proces odczytujący dane z dysku jest procesem współpracującym, czy niezależnym?



# Zastosowania komunikacji międzyprocesowej

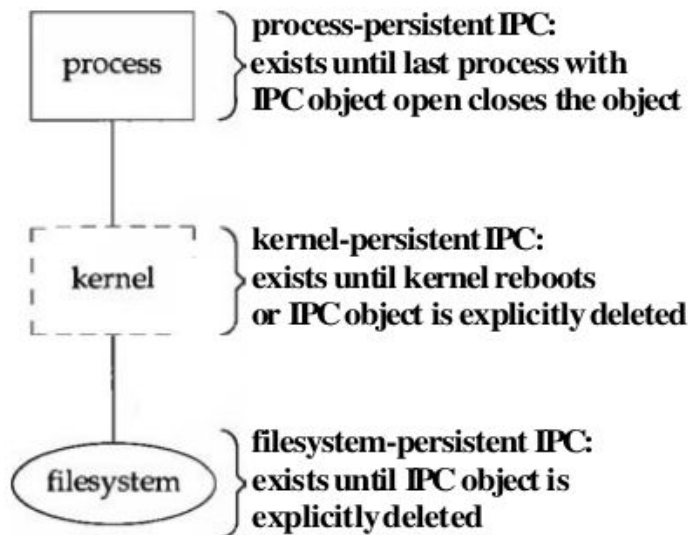
- Współdzielenie informacji
- Przyspieszenie obliczeń
- Modularność oprogramowania

# Trzy metody dzielenia informacji



Źródło: R. Stevens, Unix Network Programming - IPC

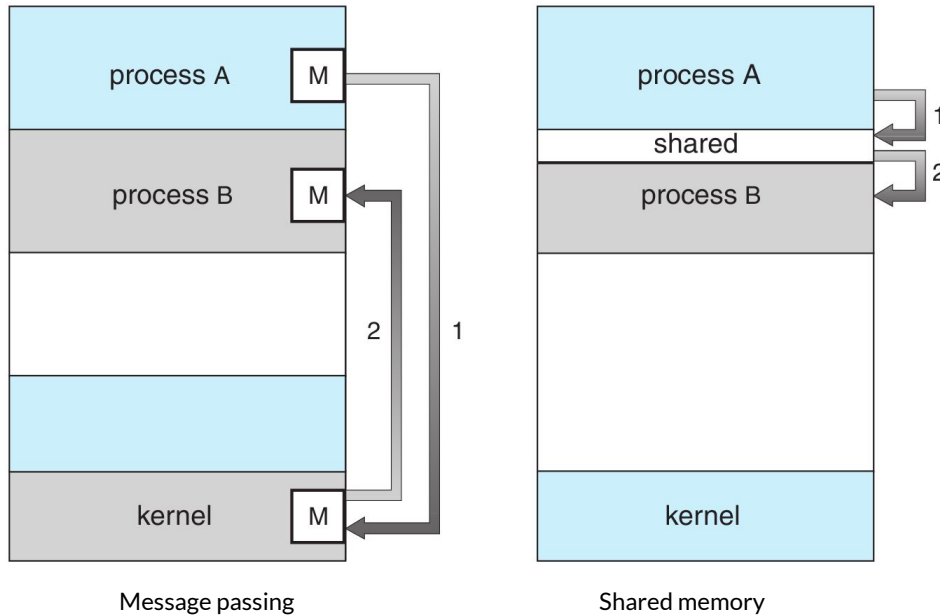
# Trwałość obiektów IPC



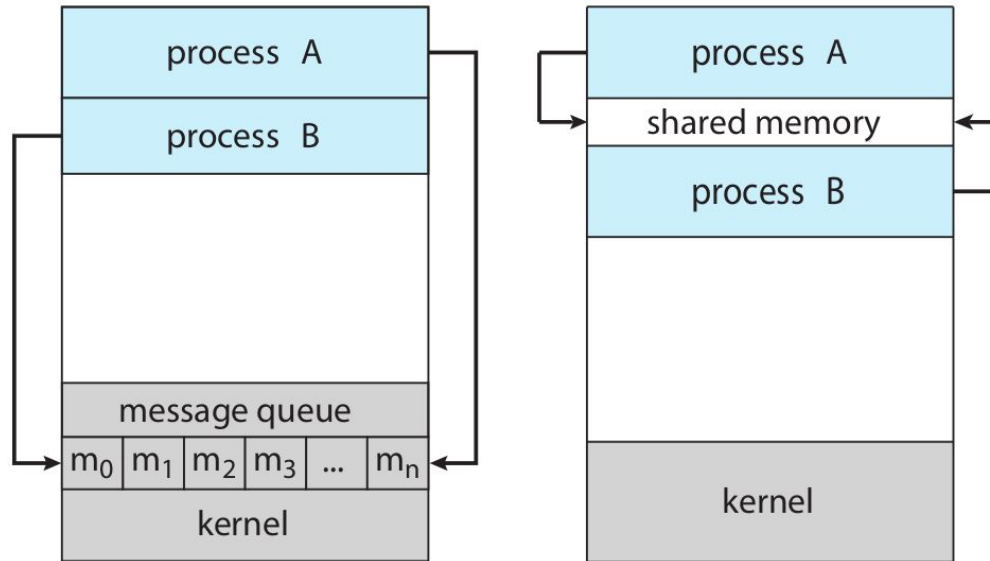
Type of IPC	Persistence
Pipe FIFO	process process
Pcsix mutex Posix condition variable Posix read-write lock fcntl record locking	process process process process
Posix message queue Pcsix named semaphore Pcsix memory-based semaphore Posix shared memory	kernel kernel process kernel
System V message queue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

Źródło: R. Stevens, Unix Network Programming - IPC

# Modele komunikacji międzyprocesowej (było!)



# Modele komunikacji międzyprocesowej (nowe!)



Message passing

Shared memory



# Shared memory (było!)





# Message-Passing Systems

Message-Passing Systems (systemy przekazywania wiadomości):

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



# Communication link

- *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:
  - Komunikacji pośredniej i bezpośredniej.
  - Komunikacji synchronicznej i asynchronicznej.
  - Buforowanie automatyczne lub na żądanie.



# Komunikacja bezpośrednia

- **Komunikacja bezpośrednia** (ang. *direct communication*) - każdy proces, który uczestniczy w komunikacji musi bezpośrednio wskazać nadawcę, czy odbiorcę:
  - `send(P, message)` - wyślij wiadomość do procesu P.
  - `receive(Q, message)` - odbierz wiadomość od procesu Q.
- **Własności:**
  - Link jest zestawiany automatycznie, a procesy muszą tylko znać swoją nazwę.
  - Link jest zestawiany dokładnie między dwoma procesami.
  - Między każdą parą komunikujących się procesów zestawiany jest dokładnie jeden link.
  - Występuje symetria w adresacji (P, Q), choć spotykane są warianty asymetryczne (np. P i id).



# Komunikacja pośrednia

- **Komunikacja pośrednia** (ang. *indirect communication*) - procesy komunikują się przez skrzynki (ang. *mailbox*) lub porty identyfikowane np. przez liczby całkowite:
  - `send(A, message)` - wyślij wiadomość do skrzynki A.
  - `receive(A, message)` - odbierz wiadomość ze skrzynki A.
- **Własności:**
  - Link jest zestawiany między parą współdzielących skrzynkę procesów.
  - Link jest może zostać zestawiony przez większą liczbę procesów.
  - Między każdą parą komunikujących się procesów mogą istnieć różne linki komunikacyjne.
- **System operacyjny musi obsłużyć następujące mechanizmy:**
  - Utworzenie skrzynki.
  - Wyśłanie i odebranie wiadomości do i ze skrzynki.
  - Usunięcie skrzynki.

(Rysunek na tablicy!)



# Synchronizacja

- Komunikacja synchroniczna, blokująca:
  - Blokujące wysyłanie - proces wysyłający jest zablokowany na wysłaniu, aż wysłana wiadomość zostanie odebrana przez proces odbierający lub skrzynkę.
  - Blokujący odbiór - odbiorca zostaje zablokowany do czasu otrzymania wiadomości.
- Komunikacja asynchroniczna, nieblokująca
  - Nieblokujące wysyłanie - proces wysyłający wysyła wiadomość i nie jest blokowany na metodzie wysyłającej.
  - Nieblokujący odbiór - odbiorca otrzymuje gotową wiadomość lub informację o braku jej dostępności.

W przypadku blokującego nadawcy i odbiorcy mamy do czynienia z ang. *Rendezvous*.

(Rysunek na tablicy!)



# Buforowanie

- Pojemność zerowa (ang. *zero capacity*) - 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.
- Ograniczona pojemność (ang. *bounded capacity*) - 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).
- Nieograniczona pojemność (ang. *unbounded capacity*) - długość kolejki jest potencjalnie nieskończona (nadawca nigdy nie jest blokowany).

(Rysunek na tablicy!)



# POSIX Shared Memory

- Tworzenie dowiązania do wspólnej przestrzeni w pamięci:
  - `int fd = shm_open(name, O_CREAT | O_RDWR, 0666);` // pisanie i czytanie
  - `int fd = shm_open(name, O_RDONLY, 0666);` // tylko czytanie
- 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:
  - `sprintf(ptr, "%s", message);` // umieszczenie komunikatu w miejscu 'ptr'
  - `ptr += strlen(message);` // przesunięcie wskazania w pamięci
- Czytanie z pamięci dzielonej:
  - `printf("%s", (char *)ptr);`
- Usunięcie obiektu współdzielonego:
  - `shm_unlink(name);`



# POSIX Shared Memory

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





## System V IPC

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



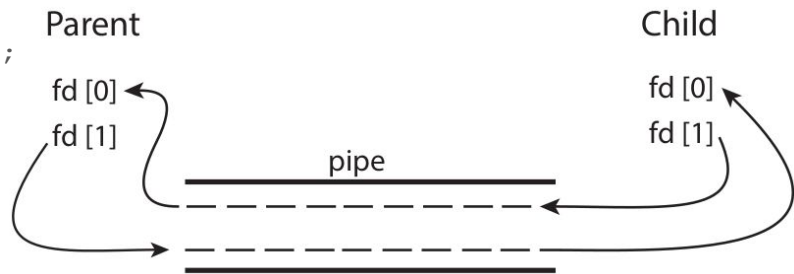
# Potok (ang. *pipe*)

- Jedne z pierwszych metod IPC, jedna z najprostszych form komunikacji.
- Przy projektowaniu potoków należy uwzględnić cztery kwestie:
  - Czy potok pozwala na dwukierunkową (ang. *bidirectional*), czy jest to jednokierunkowa (ang. *unidirectional*) komunikacja ?
  - 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) ?
  - Czy jest relacja między komunikującymi się procesami (np. rodzic - dziecko) ?
  - Czy potoki mogą komunikować się poprzez sieć, czy tylko między procesami na tej samej maszynie ?

# Potok zwykły (ang. *ordinary pipe*)

- Utworzenie potoku:
  - `int fd[2];`
  - `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);`
- Potok w praktyce:
  - `ls | less`
  - `cat file.txt | wc -l`

( Przedyskutować efekt użycia `fork()` )



Źródło: A. Silberschatz, *Operating Systems Concepts Essentials*



# Gniazda i komunikacja sieciowa

Osobny wykład.

