**Synchronizacja** to wypełnienie ograniczeń dotyczących kolejności wykonywania pewnych akcji przez procesy.
**Komunikacja** to przekazywanie informacji od jednego procesu do innego.
Wymiana danych jest zwykle oparta na **współdzieleniu zmiennych** albo **przekazywaniu wiadomości.**

# Proces

## Definicja

to przestrzeń adresowa i pojedynczy wątek sterujący, który działa w tej przestrzeni, oraz potrzebne do tego zasoby systemowe.

Minimalne zasoby do wykonywania się procesu to:
– Procesor
– Pamięć
– Urządzenia wejścia-wyjścia

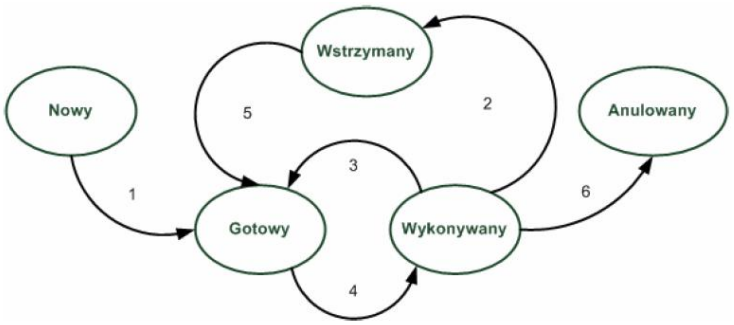
W systemach wielozadaniowych w istocie każda instancja działającego programu jest procesem

## Uruchomienie procesu

1) Podczas uruchomienia programu w OS (systemie operacyjnym) następuje przeniesienie kodu programu z nośnika do pamięci operacyjnej
2) „Otoczenie" kodu programu pewnymi strukturami danych identyfikującymi go na czas wykonania i zapewniającymi ochronę – blok kontrolny procesu
3) Powiązanie programu z zasobami, z których będzie korzystał
4) Zasobami są: procesor, pamięć, system plików, urządzenia we-wy.

Tak przygotowany program staje się ***procesem*** i jest gotowy do wykonania.

## Stany procesu




1) Utworzenie procesu
2) Żądanie zasobu, który jest niedostępny
3) Przerwanie / wywłaszczenie lub proces zwolnił procesor dobrowolnie
4) Wybranie procesu do wykonania
5) Potrzebny zasób został zwolniony
6) Zakończenie procesu

## Dane powiązane z procesem

Segment kodu (instrukcje), segment danych (statyczne dane), segment stosu (zmienne chwilowe), segment sterty (zmienne chwilowe jawnie alokowane i zwalniane), blok kontrolny procesu (dane, które o procesie przechowuje OS).

### Drzewo procesów

Procesy tworzą drzewo. Każdy proces ma jednego rodzica (jeden proces-rodzic) oraz może mieć wiele dzieci (wiele procesów-dzieci).

### Duplikowanie procesów

Wywołanie systemowe tworzy identyczny proces potomny co macierzysty – posiada unikatowy PID a PPID jest ustawiony na proces wywołujący (macierzysty). Nowy proces dziedziczy po macierzystym przestrzeń danych (zmienne) oraz otwarte deskryptory i strumienie katalogowe.

Wywołanie `fork()`:
- w procesie macierzystym zwraca PID nowego procesu
- w procesie potomnym zwraca 0

Funkcja `wait()` – powoduje zablokowanie procesu macierzystego aby poczekał, aż skończy się proces potomny (jeśli potomny skończy się wcześniej `wait()` nic nie blokuje).

# Potok

### Definicja

Przepływ danych z jednego procesu do innego. Można powiedzieć, że potoki łączą wyjście jednego procesu z wejściem innego.

### Funkcja **`pipe()`**

- udostępnia mechanizm przekazywania danych między programami, który nie wymaga uruchomienia powłoki w celu interpretacji żądanego polecenie
- do `pipe` przekazywana jest tablica dwóch liczb całkowitych będących deskryptorami plików, wszystkie dane zapisane do `file_descriptor[1]` mogą być odczytane w `file_descriptor[0]`, dane przetwarzane według FIFO
- przesyłanie danych musi się odbyć za pomocą funkcji `read` i `write` (nie `fread`, `fwrite`).

### Potoki nazwane

Umożliwiają komunikację pomiędzy procesami, które nie są ze sobą spokrewnione. Są specjalnym rodzajem pliku występującym w OS, ale zachowującym się jak potok nienazwany.

Różnica pomiędzy potokiem nienazwanym a nazwanym polega na tym, że potok nazwany ma dowiązanie w systemie plików i może być identyfikowany przez nazwę. Potok nienazwany nie posiada dowiązania i istnieje tak długo, jak długo jest otwarty. Po zamknięciu wszystkich deskryptorów związanych z potokiem nienazwanym potok przestaje istnieć.

# Wątek

### Definicja

Sekwencja działań, która może wykonywać się równolegle z innymi sekwencjami działań w kontekście danego procesu (programu).

### Różnica wątek a proces

W jednym procesie może istnieć wiele wątków. Różnica między zwykłym procesem a wątkiem polega na współdzieleniu przez wszystkie wątki działające w danym procesie przestrzeni adresowej oraz wszystkich innych struktur systemowych (np. listy otwartych plików, gniazd itp.) – z kolei procesy posiadają niezależne zasoby. Wątki wymagają mniej zasobów do działania i też mniejszy jest czas ich tworzenia. Dzięki współdzieleniu przestrzeni adresowej (pamięci) wątki jednego zadania mogą się między sobą komunikować w bardzo łatwy sposób, niewymagający pomocy ze strony systemu operacyjnego.

### Zalety stosowania wątków

- Mniejszy nakład obliczeniowy ze strony OS
- Tworzenie aplikacji z wieloma wątkami działającymi na wspólnych danych lub zasobach
- Możliwość wydzielenia w programie osobnych wątków do pobierania danych, wysyłania i przetwarzania
- Przełączanie między wątkami wymaga znacznie mniejszego nakładu obliczeniowego od OS

### Wady stosowania wątków

- Wymaga precyzyjnego projektowania, ponieważ łatwo o nieprzewidywalne wyniki
- Utrudnione śledzenie takich programów
- Aplikacja wielowątkowa na jednoprocesorowym komputerze nie musi działać szybciej

# Semafor

### Definicja

Obiekt OS, z którym związany jest licznik L zasobu przyjmujący wartości nieujemne. Na semaforze zdefiniowane są atomowe operacje `sem_init`, `sem_wait`, `sem_post`.

Chroniona zmienna lub abstrakcyjny typ danych, który stanowi klasyczną metodę kontroli dostępu przez wiele procesów do wspólnego zasobu w środowisku programowania równoległego.



### Rodzaje semaforów

1) Semafor ze zbiorem procesów oczekujących – nie jest określony, który z procesów oczekujących ma zostać zwolniony – **nie przeciwdziała zagłodzeniu**
2) Semafor z kolejką procesów oczekujących – procesy oczekujące na semaforze umieszczone są w kolejce FIFO

Inny podział

1) Semafor binarny – może mieć dwa stany – podniesiony i opuszczony. Wielokrotne podnoszenie takiego semafora nie zmieni jego stanu — skutkiem będzie stan otwarcia.
2) Semafor ogólny (zliczający) – pamięta liczbę operacji podniesienia. Przy wartości początkowej 0 można zatem bez blokowania procesu wykonać tyle operacji opuszczenia semafora, ile razy został on wcześniej podniesiony. Stąd określenie — semafor zliczający.

# Mutex

Algorytmy wzajemnego wykluczania (w skrócie często nazywane **mutex**, z ang. mutual exclusion) są używane w przetwarzaniu współbieżnym w celu uniknięcia równoczesnego użycia wspólnego zasobu (np. zmiennej globalnej) przez różne wątki/procesy w częściach kodu zwanych **sekcjami krytycznymi**. Sekcja krytyczna jest fragmentem kodu, w którym wątki (lub procesy) odwołują się do wspólnego zasobu. Sama w sobie nie jest ani mechanizmem, ani algorytmem wzajemnego wykluczania.

### Semafor a Mutex

Pierwsza różnica to fakt, że semafor może dopuścić kilka wątków naraz. W konstruktorze podajemy ile wątków może uzyskać dostęp jednocześnie do danej sekcji. Mutex jest zawsze binarny i dopuszcza wyłącznie jeden wątek.
Jaka jest różnica między semaforem binarnym a mutexem?
Mutex jest bardziej zaawansowanym tworem. Przede wszystkim zapamiętuje, z którego wątku została założona blokada.

# Producent i konsument

Problem producenta i konsumenta to klasyczny informatyczny problem synchronizacji. W problemie występują dwa rodzaje procesów: producent i konsument, którzy dzielą wspólny zasób - bufor dla produkowanych (konsumowanych) jednostek. Zadaniem producenta jest wytworzenie produktu, umieszczenie go w buforze i rozpoczęcie pracy od nowa. W tym samym

czasie konsument ma pobrać produkt z bufora. Problemem jest taka synchronizacja procesów, żeby producent nie dodawał nowych jednostek gdy bufor jest pełny, a konsument nie pobierał gdy bufor jest pusty.

Rozwiązaniem dla producenta jest uśpienie procesu w momencie gdy bufor jest pełny. Pierwszy konsument, który pobierze element z bufora budzi proces producenta, który uzupełnia bufor. W analogiczny sposób usypiany jest konsument próbujący pobrać z pustego bufora. Pierwszy producent, po dodaniu nowego produktu umożliwi dalsze działanie konsumentowi. Rozwiązanie wykorzystuje komunikację międzyprocesową z użyciem semaforów. Nieprawidłowe rozwiązanie może skutkować zakleszczeniem.

```
semaphore pełny = 0
semaphore pusty = ROZMIAR_BUFORA

procedure producent() {
    while (true) {
        produkt = produkuj()
        down(pusty)
        dodajProduktDoBufora(produkt)
        up(pełny)
    }
 }

procedure konsument() {
    while (true) {
        down(pełny)
        produkt = pobierzProduktZBufora()
        up(pusty)
        użyjProdukt(produkt)
    }
}
```

# Czytelnicy i pisarze

W systemie działa C>0 procesów, które odczytują pewne dane oraz P>0 procesów, które zapisują te dane. Procesy zapisujące będziemy nazywać pisarzami, a procesy odczytujące --- czytelnikami, zaś moment, w którym procesy mają dostęp do danych, będziemy nazywać pobytem w czytelni.
Zauważmy, że jednocześnie wiele procesów może odczytywać dane. Jednak jeśli ktoś chce te dane zmodyfikować, to rozsądnie jest zablokować dostęp do tych danych dla wszystkich innych


procesów na czas zapisu. Zapobiegnie to odczytaniu niespójnych informacji (na przykład danych częściowo tylko zmodyfikowanych).

Należy tak napisać protokoły wstępne i końcowe poszczególnych procesów, aby:

1) Wielu czytelników powinno mieć jednocześnie dostęp do czytelni.
2) Jeśli w czytelni przebywa pisarz, to nikt inny w tym czasie nie pisze ani nie czyta.
3) Każdy czytelnik, który chce odczytać dane, w końcu je odczyta.
4) Każdy pisarz, który chce zmodyfikować dane, w końcu je zapisze.

### Warianty rozwiązania

1) Wariant faworyzujący czytelników

Czytelnicy nie mają obowiązku czekania na otrzymanie dostępu do zasobu, jeśli w danym momencie nie otrzymał go pisarz. Ponieważ pisarz może otrzymać tylko dostęp wyłączny, musi czekać na opuszczenie zasobu przez wszystkie inne procesy. Jeśli czytelnicy przybywają odpowiednio szybko, może dojść do zagłodzenia pisarza: w tej sytuacji będzie on w nieskończoność czekał na zwolnienie zasobu przez wciąż napływających nowych czytelników.

2) Wariant faworyzujący pisarzy

Czytelnicy nie mogą otrzymać dostępu do zasobu, jeżeli oczekuje na niego pisarz. W tej sytuacji oczekujący pisarz otrzymuje dostęp najwcześniej, jak to jest możliwe, czyli zaraz po opuszczeniu zasobu przez ostatni proces, który przybył przed nim. W tym wariancie może dojść do zagłodzenia oczekujących czytelników.

3) Inne warianty

Inne rozwiązania problemu zakładają między innymi równoczesne wyeliminowanie możliwości zagłodzenia obu typów procesów, np. poprzez zastosowanie kolejki FIFO.

# Blokada

Zadanie (proces lub wątek) przed uzyskaniem dostępu do zasobu musi uzyskać dla siebie blokadę tego zasobu. Po zakończeniu wykorzystywania zasobu, zadanie musi blokadę zwolnić, aby udostępnić zasób innym zadaniom.

**Zajęcie blokady** - wątek wykonujący funkcję blokuje się gdy blokada jest zajęta (dla blokady nieblokującej to nie blokuje wątku). Gdy nie jest zajęta to zajmuje blokadę.
**Zwolnienie blokady** - Jeżeli jest to
ostatnia blokada i istnieją wątki czekające na jej zwolnienie to jeden z

nich zostanie odblokowany. Wybór wątku do zwolnienia zależy to od implementacji.

# Impas / zakleszczenie

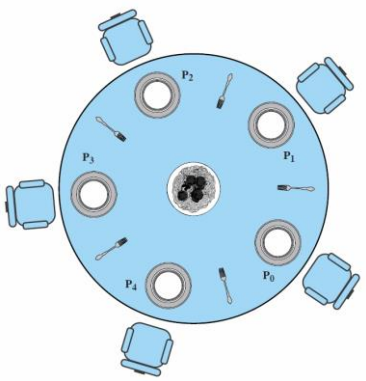
### Definicja

Trwałe zablokowanie zestawu procesów, które rywalizują o zasoby lub komunikują się ze sobą nawzajem.

Do impasu dochodzi, każdy proces zestawu jest zablokowany i oczekuje na zdarzenie (zazwyczaj na zwolnienie żądanego zasobu), które może zaistnieć tylko, jeśli zostanie zainicjowane przez inny proces z zestawu procesów.

Impas jest stanem trwałym, ponieważ żadne ze zdarzeń nigdy nie zachodzi.

W przeciwieństwie do innych problemów współbieżności, nie istnieje skuteczne rozwiązanie takiej sytuacji.




## Pięciu filozofów

Pięciu filozofów siedzi przy okrągłym stole. Przed każdym stoi talerz. Między talerzami leżą widelce. Pośrodku stołu znajduje się półmisek z rybą. Każdy filozof myśli. Gdy zgłodnieje sięga po widelce znajdujące się po jego prawej i lewej stronie, po czym rozpoczyna posiłek. Gdy się już naje, odkłada widelce i ponownie oddaje się myśleniu.

Należy tak napisać protokoły wstępne i końcowe, aby:

1) Jednocześnie tym samym widelcem jadł co najwyżej jeden filozof.
2) Każdy filozof jadł zawsze dwoma (i zawsze tymi, które leżą przy jego talerzu) widelcami.
3) Żaden filozof nie umarł z głodu.

### Możliwe rozwiązania

1) Rozwiązanie przy pomocy kelnera

   Filozofowie będą pytać go o pozwolenie przed wzięciem widelca. Ponieważ kelner jest świadomy, które widelce są aktualnie w użyciu, może nimi rozporządzać zapobiegając zakleszczeniom.

2) Rozwiązanie przy użyciu hierarchii zasobów

   Inne proste rozwiązanie jest osiągalne poprzez częściowe uporządkowanie lub ustalenie hierarchii dla zasobów (w tym wypadku widelców) i wprowadzenie zasady, że kolejność dostępu do zasobów jest ustalona przez ów porządek, a ich zwalnianie następuje w odwrotnej kolejności, oraz że dwa zasoby niepowiązane relacją porządku nie mogą zostać użyte przez jedną jednostkę w tym samym czasie.

# Blokada wirująca

### Definicja

Wirujące blokady są środkiem zabezpieczania sekcji krytycznej. Wykorzystują jednak czekanie aktywne (tj. zużywa czas procesora, wykonując cały czas pustą pętlę) zamiast przełączenia kontekstu wątku tak jak się to dzieje w mutexach.

```
while blokada aktywna do
    {nic nie rób};
```

# Równoległość a współbieżność

### Wykonanie sekwencyjne

Poszczególne akcje procesu są wykonywane jedna po drugiej.

### Wykonanie równoległe

Kilka akcji jest wykonywanych w tym samym czasie.

### Wykonanie w przeplocie

Choć jednocześnie odbywa się wykonanie tylko jednej akcji, to jednak jest wiele czynności rozpoczętych i wykonywanych na zmianę krótkimi fragmentami.

### Wykonanie współbieżne

Kolejna akcja rozpoczyna się przed zakończeniem poprzedniej. Zauważmy, że nie mówimy nic na temat tego, czy akcje te są wykonywane w tym samym czasie czy też w przeplocie. Tak naprawdę wykonanie współbieżne jest abstrakcją równoległości i zawiera w sobie zarówno wykonania równoległe jak i wykonania w przeplocie.

# Proces a program

**Program** to obiekt statyczny – tekst wykonywanego przez proces kodu. **Proces** to obiekt dynamiczny, to wykonanie programu w pewnym środowisku lub wstrzymane wykonanie (w oczekiwaniu na jakiś zasób).

# Żywotność

### Definicja

Własność żywotności można wyrazić tak: jeśli proces chce coś zrobić, to w końcu mu się to uda.

**Zakleszczenie** jest więc globalnym brakiem żywotności. Nic się nie dzieje, system nie pracuje, oczekując na zajście zdarzenia, które nigdy nie zajdzie.

**Zagłodzenie** zaś, to lokalny brak żywotności. Sytuacja w środowisku wielozadaniowym, w której dany proces nie jest w stanie zakończyć działania, ponieważ nie ma dostępu do procesora lub innego współdzielonego zasobu.

# Sygnały

### Definicja

Sygnał to zdarzenie w systemie Linux powstałe w odpowiedzi na zaistnienie pewnych okoliczności.

Po otrzymaniu sygnału proces może podjąć określone czynności.

Mogą być też jawnie wysyłane z jednego procesu do drugiego w celu przesłania informacji lub modyfikacji pracy procesu.

# Pamięć dzielona

### Definicja

Do komunikacji między rozdzielnymi procesami można zastosować mechanizm pamięci dzielonej – specjalnie zaalokowanego obszaru pamięci w obszarze procesu, w którym zastosowana jest wydzielona konwencja nazewnicza.

Należy pamiętać, że mechanizm pamięci dzielonej nie zawiera w sobie metod synchronizacji i należy je dodać.

# Kolejki komunikatów

1) Umożliwiają przesyłanie danych pomiędzy procesami w sposób podobny do potoków.
2) Pozwalają na przesyłanie komunikatów z priorytetami
3) Pozwalają na przesyłanie komunikatów o zmiennej długości, pod warunkiem ich prawidłowego odbioru (IPC-tak, POSIX-oczekują komunikatów o stałym rozmiarze)
4) Na długość kolejki i rozmiar komunikatu są nałożone systemowe ograniczenia.

Kolejki komunikatów przypominają w swojej zasadzie działania potoki, choć nie wymagają pseudo-operacji plikowych do ich otwarcia i zamykania

# Monitory

### Definicja

Obiekt, który może być bezpiecznie używany przez kilka wątków. Metody monitora chronione są przez mutexy, przez co w dowolnym momencie czasowym z dowolnej metody może korzystać tylko jeden wątek naraz. Upraszcza to budowę obiektów, zwalniając programistę z konieczności implementacji skomplikowanych wykluczeń.

1) Zmienne i działające na nich procedury zebrane są w jednym module. Dostęp do zmiennych monitora możliwy jest tylko za pomocą procedur monitora.
2) W danej chwili tylko jeden proces wykonywać może procedury monitora. Gdy inny proces wywoła procedurę monitora będzie on zablokowany do chwili opuszczenia monitora przez pierwszy proces.
3) Istnieje możliwość wstrzymania i wznowienia procedur monitora za pomocą zmiennych warunkowych.

# Zmienna warunkowa

Zmienne warunkowe są metoda synchronizacji umożliwiająca zawieszenie działania i odstąpienie czasu procesora aż do momentu, w którym pewien warunek (umieszczony w pamięci dzielonej) nie zostanie spełniony. Najprostszymi operacjami na zmiennych warunkowych są: zasygnalizowanie spełnienia warunku i oczekiwanie na spełnienie warunku.

1) Zmienne warunkowe dostarczają nowego mechanizmu synchronizacji pomiędzy wątkami. Podczas gdy muteksy implementują synchronizację na poziomie dostępu do współdzielonych danych, zmienne warunkowe pozwalają na synchronizację na podstawie stanu pewnej zmiennej.

2) Bez zmiennych warunkowych wątki musiałyby cyklicznie monitorować stan zmiennej (w sekcji krytycznej), aby sprawdzić, czy osiągnęła ona ustaloną wartość. Podejście takie jest z założenia „zasobożerne". Zmienna warunkowa pozwala na osiągnięcie podobnego efektu bez „odpytywania".
3) Zmienną warunkową stosuje się zawsze wewnątrz sekcji krytycznej w powiązaniu z zamknięciem muteksu (mutex lock).