Systemy operacyjne III

WYKŁAD 3

Jan Kazimirski

Współbieżność

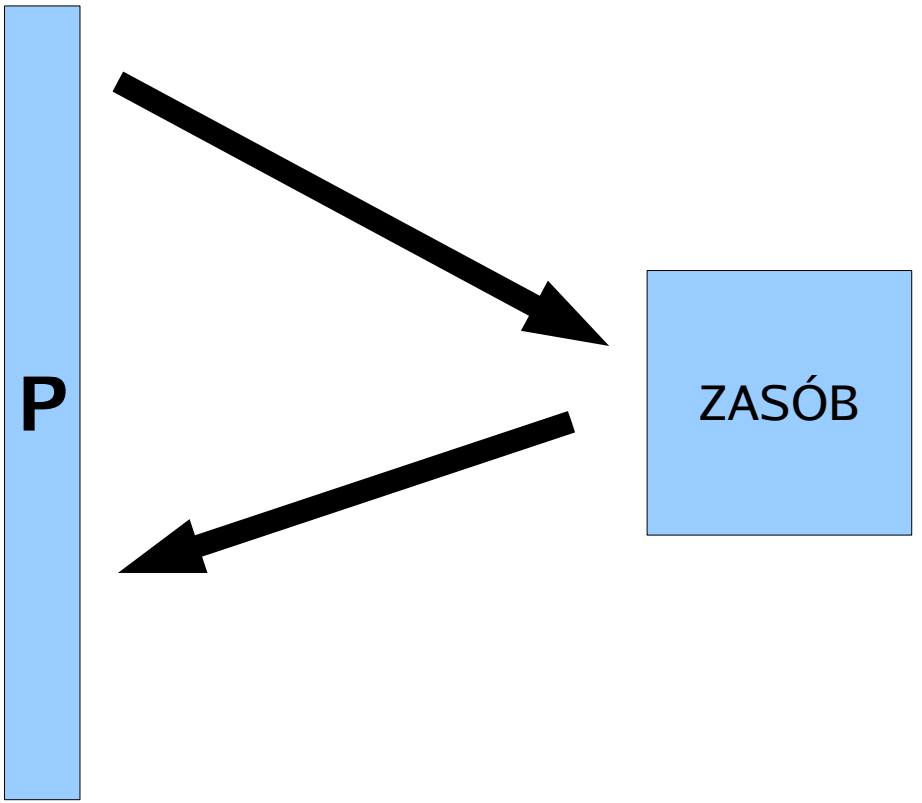
Współbieżność

- Wielozadaniowość
  - zarządzanie wieloma procesami w ramach jednego CPU
- Wieloprocesorowość
  - zarządzanie wieloma zadaniami w systemie z wieloma CPU
- Przetwarzanie rozproszone
  - zarządzanie wieloma zadaniami przetwarzanymi w architekturze rozproszonej

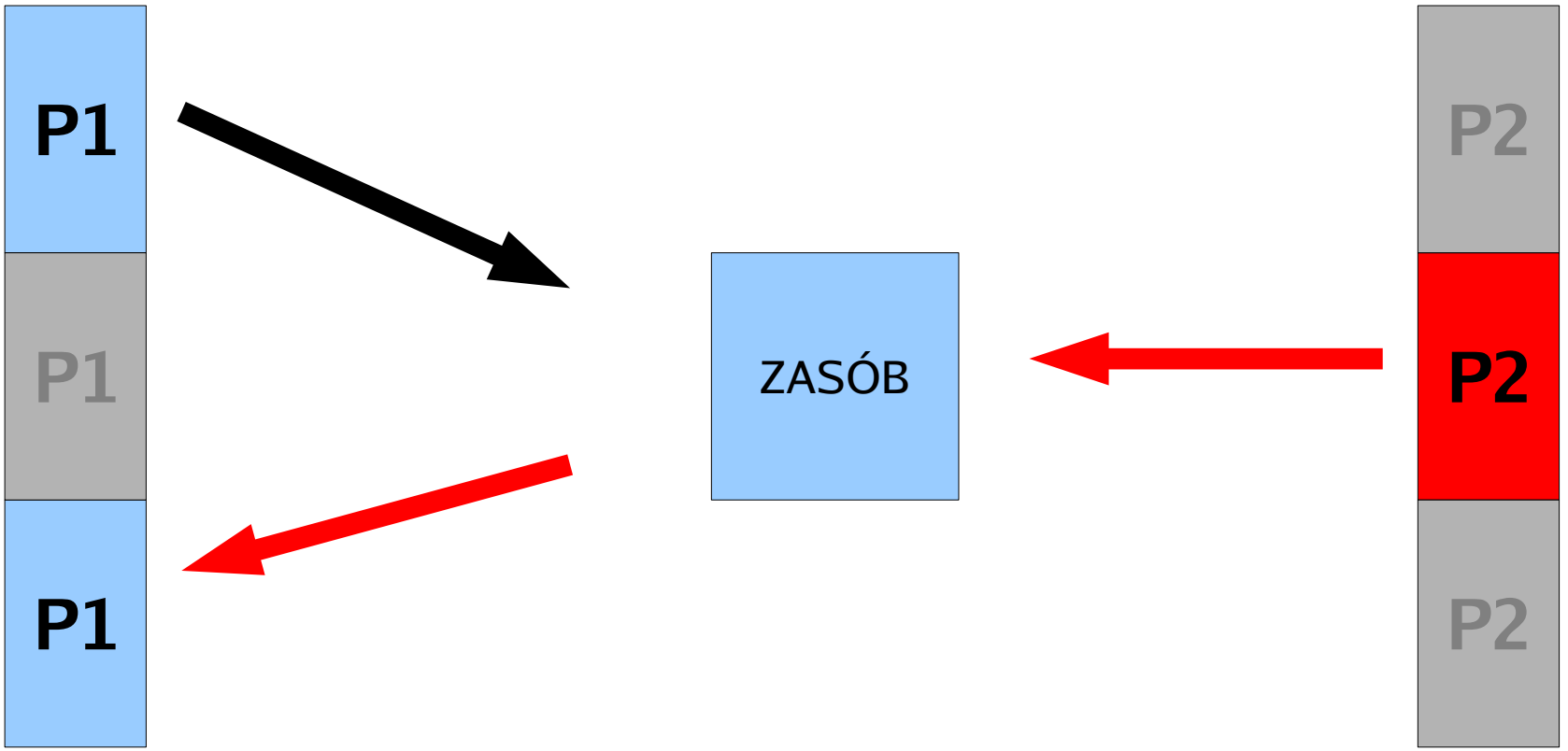
Problemy współbieżności

- Współdzielenie zasobów globalnych
  - Trudna do określenia kolejność dostępu do zasobów globalnych (np. zmienne).
- Optymalna alokacja zasobów dla procesów
  - Wydajne wykorzystywanie kanałów we/wy. Unikanie impasu.
- Problemy z testowaniem programów
  - Niedeterministyczne wykonywanie aplikacji wielowątkowych / wielozadaniowych.

Dostęp do zasobów – 1 zadanie



Dostęp do zasobów – współbieżność



Współbieżność i zmienne globalne

Program:

```
void echo() {
    chin = getchar();
    chout = chin;
    putchar(chout);
};
```

Zmienna globalna

Równoległe wykonanie echo()

P1	P2
chin = getchar();	-
-	chin = getchar();
-	chout = chin;
chout = chin;	-
putchar(chout);	-
-	putchar(chout);

Zasoby globalne w systemie wielozadaniowym

- Wymagają ochrony – zwykle przydzielane na zasadzie wyłączności
- Jeden proces korzysta z zasobu – pozostałe czekają uśpione na jego zwolnienie.
- **Problem** – system przerwań
- **Problem (system wieloprocesorowy)** – jednoczesny dostęp do zasobu.

Interakcja procesów

- Procesy w pełnej izolacji (nieświadome siebie nawzajem) – rywalizacja o zasoby
- Procesy pośrednio świadome siebie nawzajem – współpraca przez współdzielenie
- Procesy bezpośrednio świadome siebie nawzajem – możliwa komunikacja.

Rywalizacja o zasoby

- Dwa procesy próbują uzyskać dostęp do zasobu
- Procesy nie wiedzą o sobie, są od siebie odizolowane
- Przegrany proces zostanie spowolniony (musi czekać na zasób), być może nigdy go nie otrzyma.

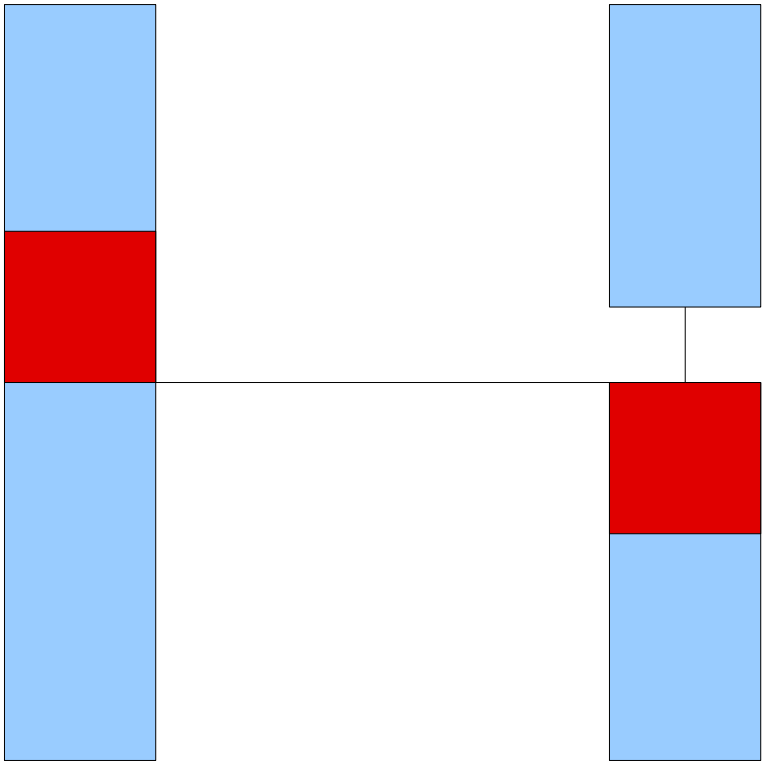
Wzajemne wykluczenie

- Kilka procesów próbuje uzyskać dostęp do zasobu – **rywalizacja o zasób**.
- Procesy są od siebie odizolowane, nic o sobie nie wiedzą.
- Tylko jeden z procesów otrzyma zasób, a pozostałe muszą czekać – **wzajemne wykluczenie**.

Wzajemne wykluczenie c.d.

- Niepodzielny zasób wymagający dostępu „na wyłączność” - **zasób krytyczny**.
- Fragment kodu realizujący dostęp do zasobu krytycznego – **sekcja krytyczna**.
- Mechanizm wzajemnego wykluczania zapewnia że tylko jeden z procesów wykonuje swoją sekcję krytyczną dla danego zasobu.

Sekcja krytyczna



Sekcja krytyczna c.d.

Program:

```
void echo() {
    chin = getchar();
    chout = chin;
    putchar(chout);
};
```

Zmienna globalna

Równoległe wykonanie echo()

P1	P2
cs.lock() ;	-
chin = getchar();	-
-	cs.lock();
chout = chin;	-
putchar(chout);	-
cs.unlock();	-
-	chin = getchar();
-	chout = chin;
-	putchar(chout);
-	cs.unlock();

Impas

P1	P2
...	...
R1.lock();	...
...	**R2.lock();**
...	...
R2.lock();	...
	R1.lock();

- Dwa procesy P1 i P2, oraz dwa zasoby R1 i R2.
- Każdy proces wymaga dostępu do obu zasobów
- P1 rezerwuje R1, a P2 rezerwuje R2
- Każdy proces czeka na zasób zarezerwowany przez drugi z procesów

Zagłodzenie

P1	P2	P3
...	...	...
R.lock();	**R.lock();**	**R.lock();**
...		
R.unlock();		
...	...	
R.lock();	...	
	R.unlock();	
...	...	
...	**R.lock();**	
R.unlock();		
...	...	

Procesy P1,P2,P3 wymagają dostępu do zasobu R

P1 posiada zasób, P2 i P3 czekają

P1 wychodzi z sekcji krytycznej, zasób otrzymuje P2.

P1 ponownie zgłasza zapotrzebowanie na zasób

P2 zwalnia zasób, system przydziela go do P1. P3 czeka

P3 może nigdy nie otrzymać zasobu, pomimo braku impasu

Algorytmy wzajemnego wykluczania - założenia

- Musi być wymuszone. Tylko jeden proces może być w swojej sekcji krytycznej (dla danego zasobu)
- Proces zatrzymany nie może przeszkadzać innym procesom
- Proces oczekujący w końcu musi otrzymać zasób
- Pierwszy proces żądający dostępu do swojej sekcji krytycznej otrzymuje go
- Mechanizm nie może zależeć od liczby i szybkości procesów
- Proces może przebywać w sekcji krytycznej tylko określony czas

Algorytm wzajemnego wykluczania – architektura z 1 procesorem

- W architekturze z jednym procesorem w danym momencie wykonywany jest tylko jeden proces.
- Wymuszenie wzajemnego wykluczania:
  - Blokada przełączania procesów w czasie wykonywania sekcji krytycznej
  - Wyłączenie przerwań na czas wykonywania sekcji krytycznej procesu

Algorytm wzajemnego wykluczania – architektura wieloprocesorowa

- W architekturze wieloprocesorowej procesy na różnych procesorach wykonują się jednocześnie.
- Zablokowanie przerwań i przełączania procesów w sekcji krytycznej nie rozwiązuje problemu
- Wykorzystywana jest własność pamięci operacyjnej – procesor ma wyłączny dostęp do danej komórki pamięci.
- Specjalne rozkazy procesora – „odczyt i zapis” lub „odczyt i test” realizowane jako operacje atomowe (nieprzerywalne)

Wzajemne wykluczenie poprzez zmienną współdzieloną

- Procesy P1,P2...,Pn wymagają dostępu do zasobu – kontrola za pomocą zmiennej T
- T=0, zasób wolny
- P1 testuje i ustawia T – rezerwuje zasób
- Pozostałe procesy czekają na zwolnienie zasobu (**aktywne oczekiwanie**)
- P1 zwalnia zasób, T->0, zasób przejmuje kolejny proces.

Specjalne rozkazy procesora – zalety i wady

- Dowolna liczba procesów, systemy jedno- i wieloprocesorowe
- Prostota
- Obsługa wielu sekcji krytycznych
- Wykorzystanie aktywnego oczekiwania
- Możliwy stan zagłodzenia i impas.

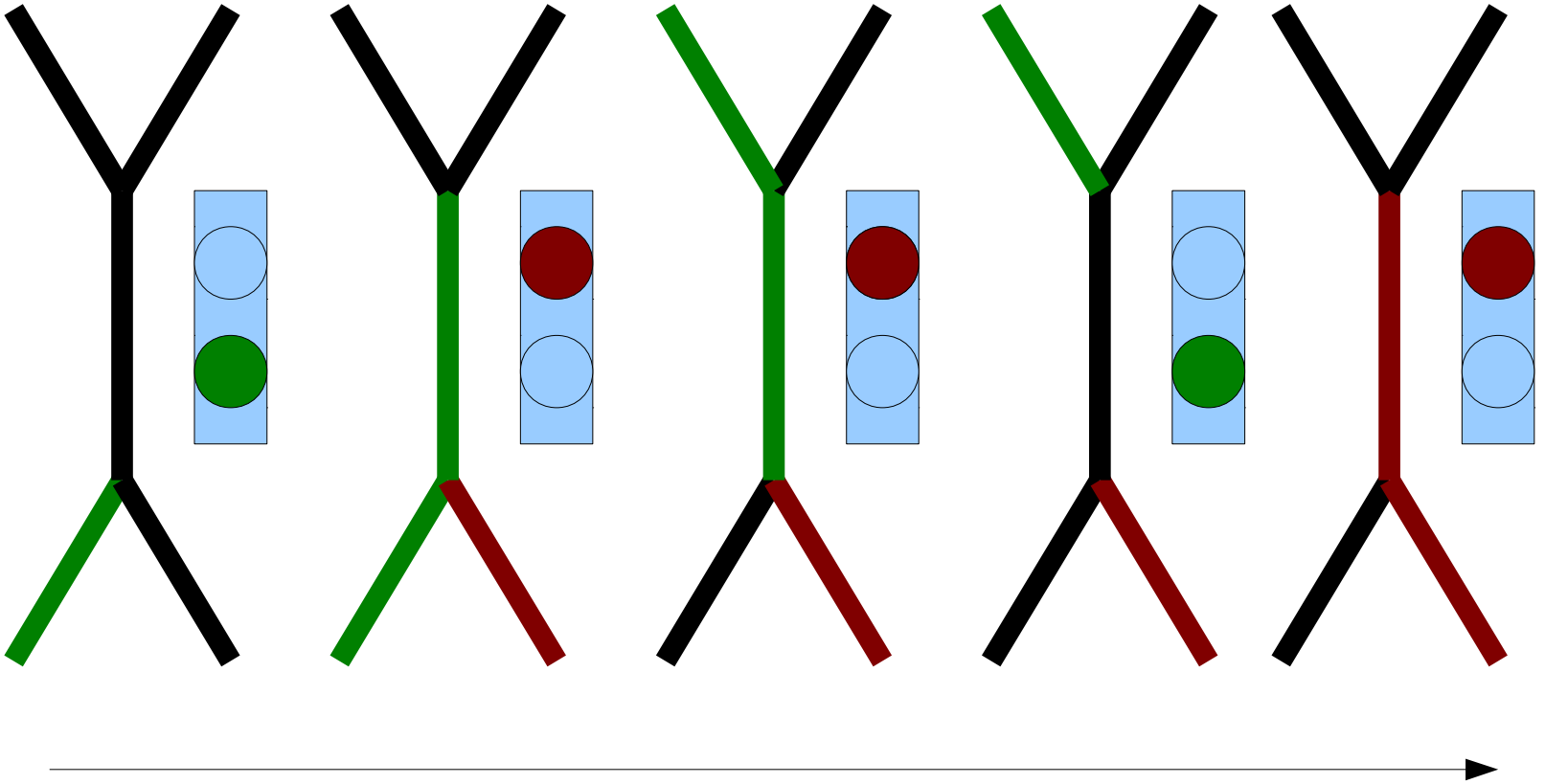
Semafor

- Zmienna całkowita ze zdefiniowanymi operacjami:
  - inicjacja liczbą dodatnią
  - **semWait** - operacja zmniejszenia wartości, wartość ujemna blokuje operację
  - **semSignal** - operacja zwiększenia wartości, jeżeli jest mniejsza lub równa 0, czekający proces jest odblokowany

Mutex

- Semafor binarny (mutex)
  - można go inicjalizować tylko na 0 lub 1
  - **semWait** – testuje semafor, wartość 0 blokuje proces, wartość jeden zmieniana jest na zero
  - **semSignal** – odblokowuje zablokowany proces lub ustawia wartość na 1

Mutex c.d.



Monitor

- Programowe rozszerzenie funkcjonalności semaforów
- Moduł składający się z procedur i danych lokalnych, oraz zapewniający że:
  - dostęp do danych lokalnych mają tylko procedury monitora
  - proces wchodzi do monitora wywołując jedną z jego procedur
  - tylko jeden proces może być wykonywany w monitorze, inne są zablokowane

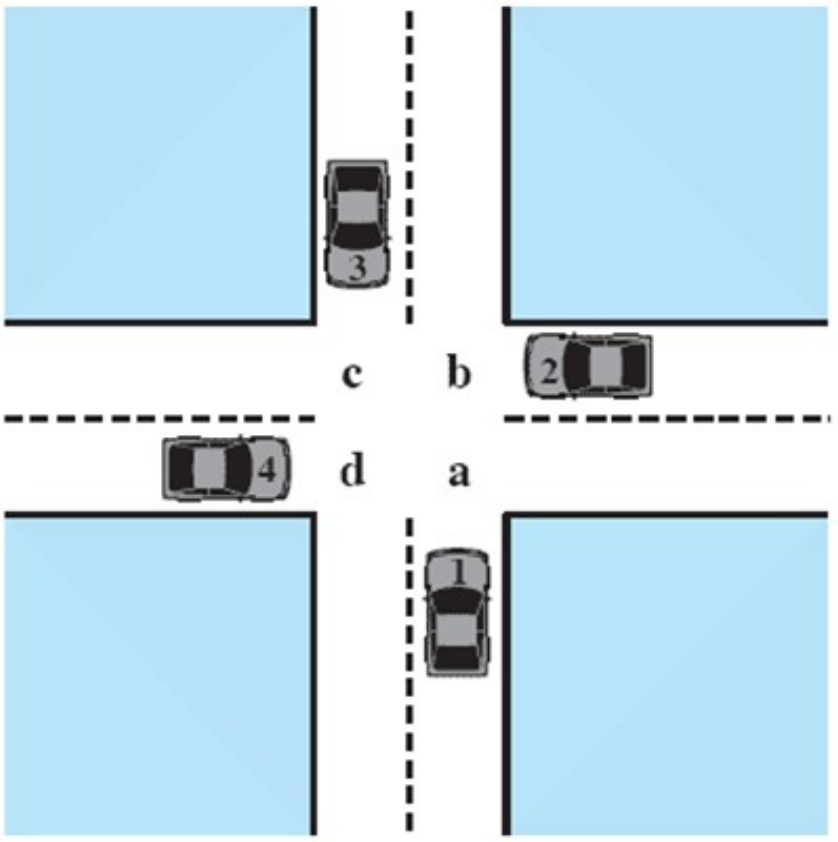
Komunikaty

- Przetwarzanie rozproszone – procesy dysponują osobnymi środowiskami wykonawczymi
- Komunikacja i synchronizacja procesów – poprzez wymianę komunikatów
- Przekazywanie komunikatów można też implementować w innych modelach przetwarzania.

Komunikaty c.d.

- Typowa implementacja: send/receive
- Synchronizacja:
  - synchroniczna wymiana komunikatów (blokująca)
  - asynchroniczna wymiana komunikatów (nieblokująca)

Impas



Stallings, „Systemy operacyjne ...”

Warunki wystąpienia impasu

- Wzajemne wykluczanie
  - Tylko jeden proces ma w danej chwili dostęp do zasobu.
- Wstrzymanie i oczekiwanie
  - Proces może trzymać rezerwację zasobu czekając na inne.
- Brak wywłaszczania
  - Nie ma możliwości odebrania procesowi zasobu
- Cykliczne oczekiwanie
  - Istnieje łańcuch procesów taki, że każdy z nich przetrzymuje zasób potrzebny innemu procesowi

Zapobieganie impasom

- Rezygnacja z wzajemnego wykluczania
  - Generalnie niepraktyczne (problem rywalizacji o zasoby).
  - Możliwość częściowej rezygnacji (np. równoległy odczyt z pliku przez kilka procesów).
- Zapobieganie przetrzymywaniu zasobów
  - Proces musi rezerwować wszystkie zasoby jednocześnie. Brak jednego zasobu powoduje zwolnienie pozostałych.
  - Problemy wydajnościowe
  - Problemy z implementacją (informacja o potrzebnych zasobach nie zawsze jest dostępna z góry).

Zapobieganie impasom

- Wywłaszczanie
  - SO może odebrać zasób danemu procesowi i przydzielić innemu.
  - Praktyczne tylko jeżeli stan zasobu można łatwo zachować i odtworzyć.
- Cykliczne oczekiwanie
  - Wymuszenie kolejności rezerwacji zasobów.
  - Problemy z wydajnością

Unikanie impasu

- Zapobieganie impasom to podejście statyczne - „Wbudowane” w projekt SO i obniżające wydajność.
- Unikanie impasu – działanie dynamiczne.
  - Decyzja o przydziale zasobu na podstawie aktualnej sytuacji.
  - Zasób nie zostanie przydzielony jeżeli może to doprowadzić do impasu.

Wykrywanie impasu

- Zapobieganie i unikanie impasu to podejścia konserwatywne. Mogą prowadzić do problemów z implementacją i wydajnością.
- Inne podejście – brak ograniczeń w przydziale zasobów i wykrywanie ewentualnego impasu.
- W przypadku zaistnienia impasu realizowana jest procedura usuwania impasu.

Algorytmy usuwania impasu

- Przerwanie wszystkich procesów w impasie.
- „Wycofanie” do ostatniego zapisanego stanu (rollback).
- Przerywanie procesów w impasie kolejno aż do usunięcia impasu.
- Wywłaszczanie procesów kolejno aż do usuniecia impasu.

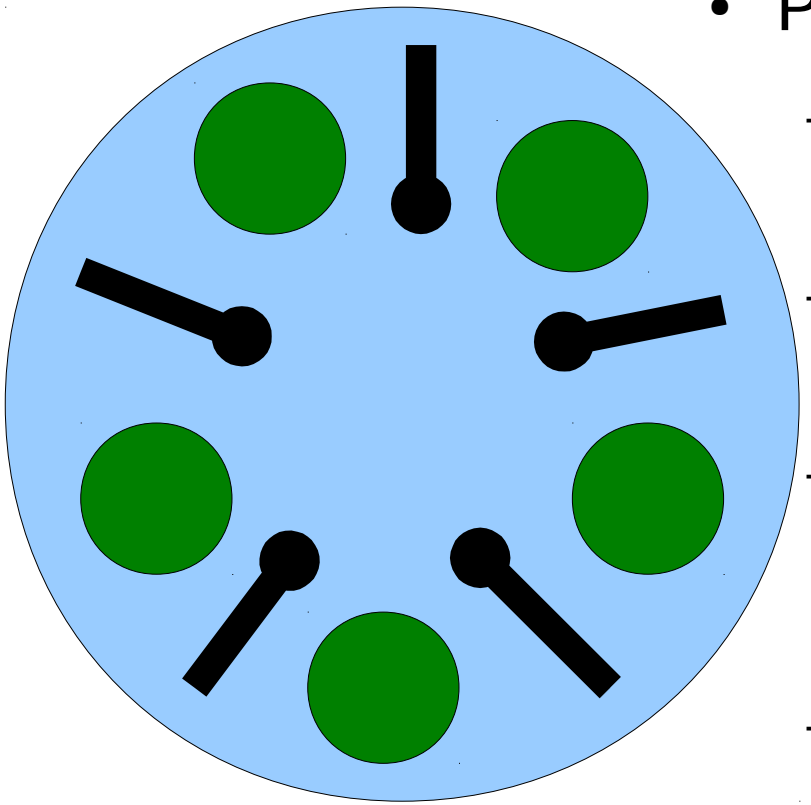
Klasyczne problemy synchronizacji

- Problem producent/konsument
  - Dwa typy procesów: producent – generuje dane, konsument – zużywa dane
  - Dane umieszczane są i pobierane z bufora
  - Problem: synchronizacja zadań w celu uniknięcia przepełnienia bufora lub próby pobrania danych z pustego bufora.

Klasyczne problemy synchronizacji c.d.

- Problem czytelników i pisarzy
  - Dwa typy procesów: czytelnicy (dostęp niemodyfikujący do zasobu) i pisarze (dostęp modyfikujący)
  - Czytelnicy mają nieograniczony dostęp, pisarze wymagają zasobu na wyłączność
  - Różne warianty: uprzywilejowani czytelnicy lub pisarze.
  - Problem: unikanie zagłodzenia czytelnika lub pisarza

Klasyczne problemy synchronizacji c.d.



- Problem ucztujących filozofów
  - Okrągły stół i naprzemiennie leżące talerze i widelce
  - Filozofowie siedzą przy stole i oddają się medytacji.
  - Co jakiś czas filozof posila się korzystając z talerza i dwóch widelców (dobieranych pojedynczo)
  - Występują problemy związane z rywalizacją o zasoby, impasem, zagłodzeniem

Mechanizmy synchronizacji w systemie UNIX/Linux

- Systemy UNIX/Linux oferują różne mechanizmy synchronizacji i komunikacji między procesami:
  - potoki
  - komunikaty
  - pamięć współdzielona
  - semafory
  - sygnały

Potok

- Bufor umożliwiający komunikację dwóch programów w modelu producent/konsument
- Potok ma bufor o określonym rozmiarze
  - Jeżeli jest miejsce w buforze to zapis jest natychmiastowy, w przeciwnym wypadku proces jest zablokowany
  - Proces czytający odbiera dane z bufora. Gdy bufor jest pusty to proces zostaje zablokowany

Komunikaty

- Bloki bajtów z dodatkowym znacznikiem typu.
- Wysyłanie i odbieranie komunikatów: *msgsnd*, *msgrcv*.
- Komunikaty przechowywane są w buforze FIFO.
- Czytanie z pustego bufora lub zapis do pełnego bufora są blokowane

Pamięć współdzielona

- Blok pamięci wirtualnej do której ma dostęp wiele procesów
- Tworzenie/dostęp: funkcja *shmget.*
- Dołączanie i odłączanie bloku pamięci dzielonej: *shmat*, *shmdt*.

Semafory

- Semafory nazwane i nienazwane (w pamięci dzielonej)
- Obsługa semaforów:
  - tworzenie: *sem_init* lub *sem_open*
  - usuwanie: *sem_destroy* lub *sem_close*
  - operacje: *sem_post*, *sem_wait*

Sygnały

- Mechanizm informujący proces o wystąpieniu zdarzenia.
- Przypomina przerwanie sprzętowe (ale bez priorytetów).
- Procesy mogą wysyłać i odbierać sygnały.
- Proces może zareagować na sygnał wywołując domyślną czynność lub czynność zdefiniowaną przez programistę.