Wykład 4

Synchronizacja procesów (i wątków) cześć I

Potrzeba synchronizacji

- Procesy wykonują się współbieżnie.
- Jeżeli w 100% są izolowane od siebie, nie ma problemu.
- Problem, jeżeli procesy komunikują się lub korzystają ze wspólnych zasobów.
- Potrzeba utrzymywania wspólnych zasobów w spójnym stanie.
- Dotyczy także wątków

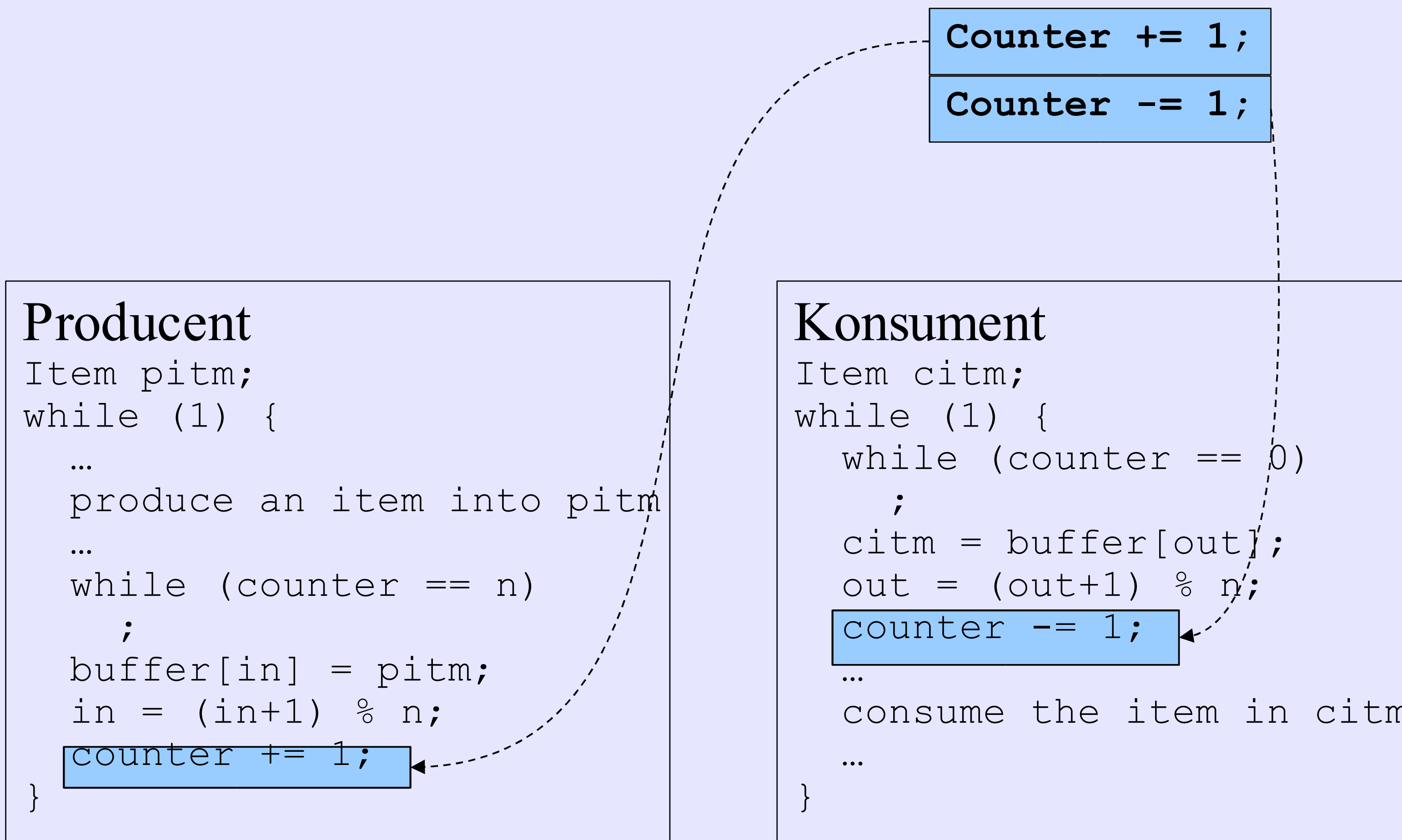
Przykład
Producent konsument z buforem cyklicznym

Zmienne wspólne

```
const int n;                 // rozmiar bufora
typedef … Item;
Item buffer[n];              // bufor
int out=0;                   // indeks konsumenta
int in = 0;                  // indeks producenta
counter = 0;                 // liczba elementów w buforze
```

- Producent umieszcza element w buforze na pozycji *in*
  - Czeka, jeżeli counter==n, tzn. bufor pełny
- Konsument pobiera element z bufora z pozycji *out*
  - Czeka, jeżeli counter==0 tzn. bufor pusty.
- Zmienne *in* oraz *out* zmieniane są zgodnie z regułą
  - $i=(i+1)\%n$
  - Wartości kolejnych indeksów do tablicy buffer
    - Jeżeli i==n-1 to *nowei*=0

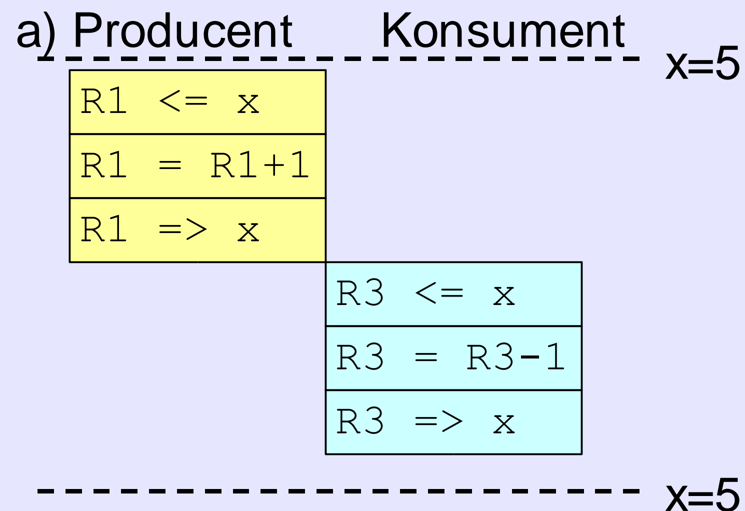
Rozwiązanie z wykorzystaniem aktywnego czekania

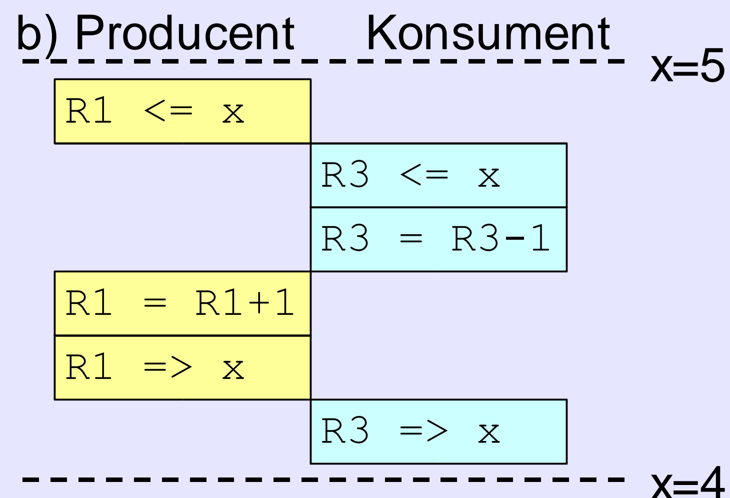


- Counter jest zmienną współdzieloną przez obydwa procesy.
- Co się może stać gdy jednocześnie obydwa procesy spróbują się do niej odwołać ?

Gdzie tkwi problem?

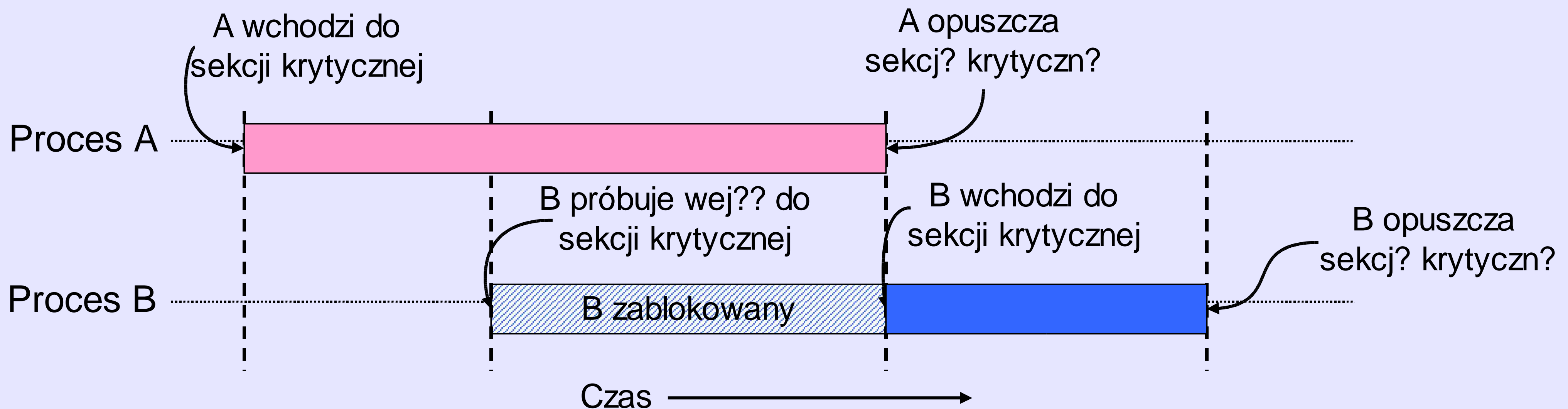
- Architektura RISC: ładuj do rejestru, zwiększ wartość,zapisz wynik.
- Niech x oznacza jest modyfikowaną zmienną *counter*.
  - Przyjmijmy, że x=5
- Rozważmy dwie możliwe kolejności wykonywania instrukcji poszczególnych procesów.
  - a) Poprawna wartość 5.
  - b) Niepoprawna wartość 4.
- Wybór jednej z tych wielkości niedeterministyczny.
- ***Sytuacja wyścigu (ang. race)***





Problem sekcji krytycznej

- Założenia
  - Proces na przemian przebywa w sekcji krytycznej albo wykonuje inne czynności
  - Proces przebywa w sekcji krytycznej przez skończony czas.
- Rozwiązanie
  - Czynności wykonywane przy wejściu do sekcji (protokół wejścia)
  - Czynności wykonywane przy wyjściu z sekcji (protokół wyjścia)



Warunki dla rozwiązania sekcji krytycznej

- Wzajemne wykluczanie.
  - W danej chwili tylko jeden proces może być w sekcji krytycznej.
- Postęp
  - Proces który nie wykonuje sekcji krytycznej nie może blokować procesów chcących wejść do sekcji.
- Ograniczone czekanie
  - Proces nie może czekać na wejście do sekcji krytycznej w nieskończoność

Rozwiązania problemu sekcji krytycznej

- Wyłącz przerwania
  - Nie działa w systemach wieloprocesorowych
  - Problematyczne w systemach z ochroną
  - Wykorzystywane do synchronizacji w obrębie jądra (zakładając jeden procesor)
- Rozwiązania z czekaniem aktywnym
  - Algorytm Petersona dla dwóch procesów
  - Algorytm piekarni dla wielu procesów.
  - Rozwiązania wykorzystujące specjalne instrukcje maszynowe np. rozkaz zamiany:
    - XCHG rejestr, pamięć
    - Architektura systemu musi zapewniać atomowe wykonanie instrukcji.
    - W systemach wieloprocesorowych nie jest to trywialne

.

Czekanie aktywne

- Marnowany jest czas procesora
  - Zmarnowany czas można by przeznaczyć na wykonanie innego procesu.
- Uzasadnione gdy:
  - Czas oczekiwania stosunkowo krótki (najlepiej krótszy od czasu przełączenia kontekstu)
  - Liczba procesów ≅ Liczba procesorów
- Przykład zastosowania jądro Linux-a w wersji SMP
  - Funkcje typu *spin_lock*
- Alternatywą do czekania aktywnego jest przejście procesu w stan zablokowany
  - Semafory
  - Monitory

Semafor (zliczający)

- Zmienna całkowita *S* na której, oprócz nadania wartości początkowej, mogą być wykonane dwie operacje
- Definicje na podstawie [BenAri89]
- Operacja P (czekaj, wait)
  - Jeżeli S>0 to S:=S-1, w przeciwnym razie wstrzymaj działanie procesu wykonującego tę operacje.
- Operacja V(sygnalizuj,signal).
  - Jeżeli są procesy wstrzymane w wyniku operacji P, to wznów jeden z nich, w przeciwnym wypadku S:=S+1.
- Operacje P i V są operacjami atomowymi.

Implementacja semafora

- Semafor to
  - Bieżąca wartość
  - Lista procesów oczekujących
- Lista najlepiej typu FIFO
  - LIFO, kolejki priorytetowe mogą prowadzić do zagłodzenia
- Sleep: realizuje przejście procesu Aktywny=>Zablokowany
- Wakeup:]Zablokowany=>Gotowy

```
class Semaphore {
  int value;
  ProcessList pl;
public:
  Semaphore(int a) {value=a;}
  void down ();
  void up ();
};

Semaphore::P() ()
{
  value -= 1;
  if (value < 0) {
    Add(this_process,pl)
    Sleep (this_process);
  }
}

Semaphore::V () {
  value += 1;
  if (value <= 0) {
    Process P=Remove(P)
    Wakeup (P);
  }
}
```

Rozwiązanie sekcji problemu sekcji krytycznej przy pomocy semaforów

```
Semaphore Sem(1);

void Process() {
    while (1) {
        Sem.P();
        // Proces wykonuje swoją sekcję krytyczną
        Sem.V()
        // Proces wykonuje pozostałe czynności
    }
}
```

- Protokół wejścia i wyjścia są trywialne, ponieważ semafory zaprojektowano jako narzędzie do rozwiązania problemu sekcji krytycznej

- Zmodyfikujmy warunki zadania, tak że w sekcji krytycznej może przebywać jednocześnie co najwyżej *K* procesów.
- ***Pytanie.*** Co należy zmienić w programie ?

Problem producent-konsument z wykorzystaniem semaforów

```
const int n;
Semaphore empty(n),full(0),mutex(1);
Item buffer[n];
```

Producent

```
int in = 0;
Item pitem;
while (1) {
  // produce an item into pitem
  empty.P();
  mutex.P();
  buffer[in] = pitem;
  in = (in+1) % n;
  mutex.V();
  full.V();
}
```

Konsument

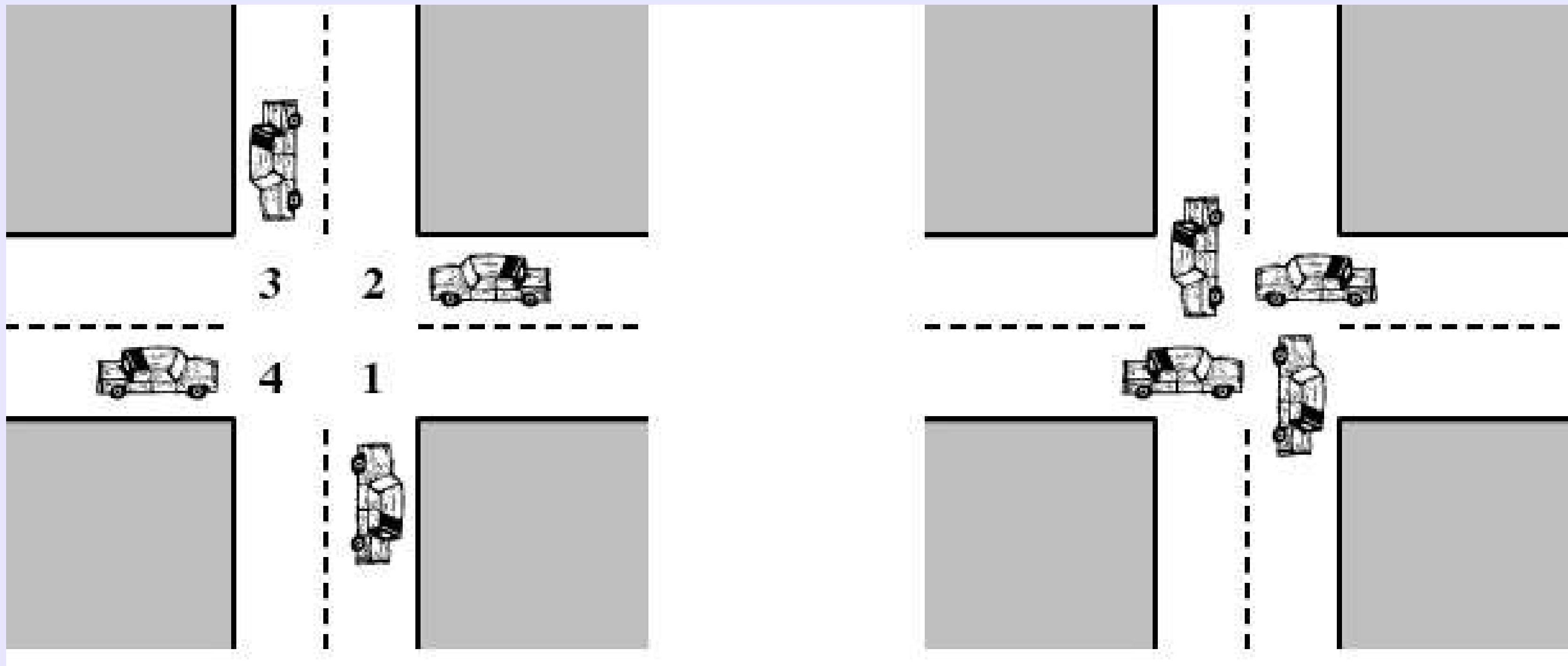
```
int out = 0;
Item citem;
while (1) {
  full.P();
  mutex.P();
  citem = buffer[out];
  out = (out+1) % n;
  mutex.V();
  empty.V();
  // consume item from citem
}
```

- Semafor *mutex* zapewnia wzajemne wykluczanie przy dostępie do zmiennych współdzielonych
- Semafor *full* gwarantuje oczekiwanie konsumenta, gdy bufor jest pusty.
- Semafor *empty* gwarantuje oczekiwanie producenta, gdy bufor jest pełny.

Semafory binarne

- Zmienna może przyjmować tylko wartość zero lub jeden
  - Operacje mają symbole PB, VB
  - Wartość jeden oznacza, że można wejść do semafora (wykonać PB)
  - Wartość zero oznacza że operacja V wstrzyma proces.
- Mogą być prostsze w implementacji od semaforów zliczających.
- Implementacje
  - Mutexy w POSIX threads.
  - W win32 mutexy noszą nazwę sekcji krytycznych
  - W Javie mutex jest związany z każdym obiektem
    - Słowo kluczowe *synchronized*.
    - Więcej o Javie przy omawianiu monitorów

Blokada (Zakleszczenie, ang. deadlock)



- Zbiór procesów jest w stanie blokady, kiedy każdy z nich czeka na zdarzenie, które może zostać spowodowane wyłącznie przez jakiś inny proces z tego zbioru.
- Samochody nie mają wstecznego biegu = Brak wywłaszczeń zasobów

Przykład blokady

- Sekwencja instrukcji prowadząca do blokady.
  - P_0 wykonał operacje A.P()
  - P_1 wykonał operacje B.P()
  - P_0 usiłuje wykonać B.P()
  - P_1 usiłuje wykonać A.P()
  - P_0 czeka na zwolnienie B przez P_1
  - P_1 czeka na zwolnienie B przez P_0
  - ***Będą czekały w nieskończoność !!!***
- Do blokady ***może*** (ale nie musi) dojść.
- ***Pytanie:*** Jak w tej sytuacji zagwarantować brak blokady ?

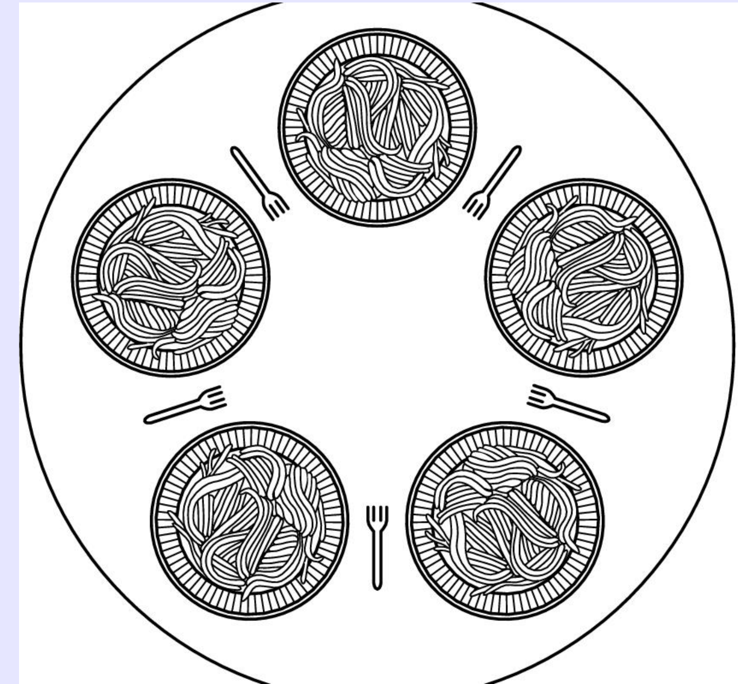
```
Semaphore A(1),B(1);
```

Proces P_0	Proces P_1
`A.P();`	`B.P();`
`B.P();`	`A.P();`
.	.
.	.
.	.
`B.V();`	`A.V();`
`A.V();`	`B.V();`

Zagłodzenie (ang. starvation)

- Proces czeka w nieskończoność, pomimo że zdarzenie na które czeka występuje. (Na zdarzenie reagują inne procesy)
- Przykład: Jednokierunkowe przejście dla pieszych, przez które w danej chwili może przechodzić co najwyżej jedna osoba.
  - Osoby czekające na przejściu tworzą kolejkę.
  - Z kolejki wybierana jest zawsze najstarsza osoba
  - Bardzo młoda osoba może czekać w nieskończoność.
- Zamiast kolejki priorytetowej należy użyć kolejki FIFO (wybieramy tę osobę, która zgłosiła się najwcześniej)

Problem pięciu filozofów



- Każdy filozof siedzi przed jednym talerzem
- Każdy filozof na przemian myśli i je
- Do jedzenia potrzebuje dwóch widelców
  - Widelec po lewej stronie talerza.
  - Widelec po prawej stronie talerza.
- W danej chwili widelec może być posiadany tylko przez jednego filozofa.
- Zadanie: Podaj kod dla procesu *i-tego* filozofa koordynujący korzystanie z widelców.

Problem czytelników i pisarzy

- Modyfikacja problemu sekcji krytycznej.
- Wprowadzamy dwie klasy procesów: *czytelników* i *pisarzy*.
- Współdzielony obiekt nazywany jest *czytelnią*.
- W danej chwili w czytelni może przebywać
  - Jeden proces pisarza i żaden czytelnik.
  - Dowolna liczba czytelników i żaden pisarz.
- Rozwiązanie *prymitywne*: Potraktować czytelnię jak obiekt wymagający wzajemnego wykluczania wszystkich typów procesów.
  - Prymitywne, ponieważ ma bardzo słabą wydajność. Jeżeli na wejście do czytelni czeka wielu czytelników i żaden pisarz to możemy wpuścić od razu wszystkich czytelników
- W literaturze opisano rozwiązania:
  - Z możliwością zagłodzenia pisarzy
  - Z możliwością zagłodzenia czytelników
  - Poprawne

Problem śpiącego fryzjera

- Jeden proces *fryzjera* i wiele procesów *klientów*.
- Współdzielone zasoby: n krzeseł w poczekalni i jedno krzesło fryzjera
- Napisz program koordynujący pracę fryzjera i klientów