1 Zakleszczenia

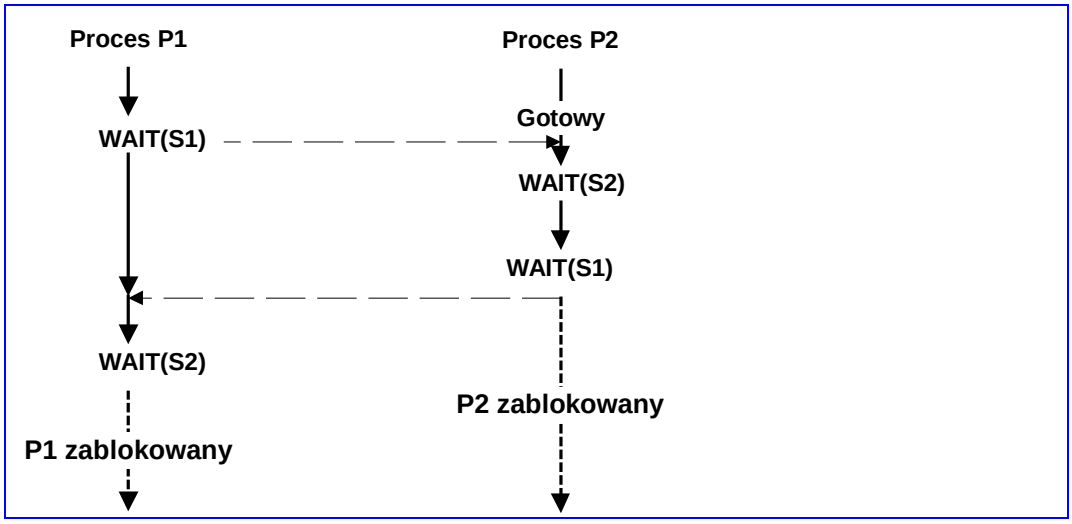
1.1 *Prosty przykład zakleszczenia (*ang. Mexican standoff*)*

W systemach w których wykonywane jest wiele współbieżnych procesów które operują na wspólnych zasobach może dojść do niezamierzonego wstrzymania pracy pewnych procesów. Mówi się że procesy mogą ulec zakleszczeniu.

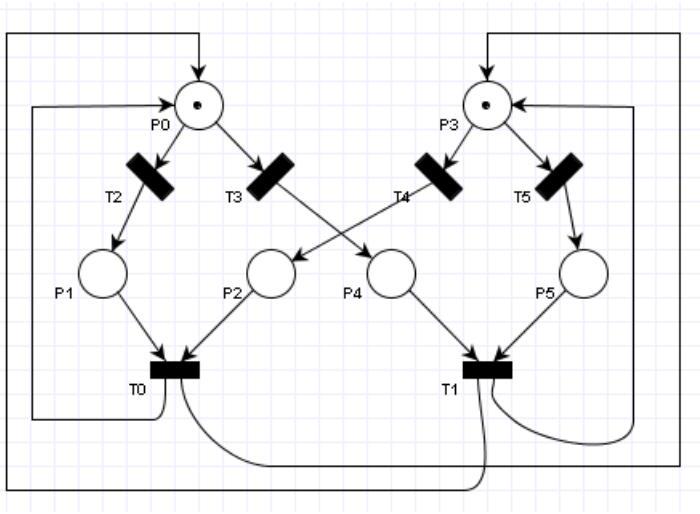
Procesy P1 i P2 aby wykonać swoje zadania potrzebują zasobów Z1 i Z2. Proces P1 najpierw próbuje zająć zasób Z1 a następnie zasób Z2. Proces P2 najpierw próbuje zająć zasób Z2 a następnie zasób Z1. Zasoby te zabezpieczone są semaforami S1 i S2.

```
sem_type S1, S1;

Proces1(void){
  do {
     ....
    sem_wait(S1);   // Zamówienie zasobu Z1
    sem_wait(S2);   // Zamówienie zasobu Z2
    uzycie(S1,S2);  // Użycie zasobów Z1 i Z2
    sem_post(S2);   // Zwolnienie zasobu Z2
    sem_post(S1);   // Zwolnienie zasobu Z1
  } while(1);
}

Proces2(void) {
  do {
     ....
     sem_wait(S2);       // Zamówienie zasobu Z2
     sem_wait(S1);       // Zamówienie zasobu Z1
     uzycie(S1,S2);      // Użycie zasobów Z1 i Z2
     sem_post(S1);       // Zwolnienie zasobu Z1
     sem_post(S2);       // Zwolnienie zasobu Z2
    } while(1);
}

main(void) {
// Inicjacja semaforów na 1
 sem_init(S1,1)
 sem_init(S2,1)
 // Utworzenie wątków P1 i P2
 …
}
```

Przykład 1-1 **Zastój meksykański – przykład systemu w którym może dojść do zakleszczenia**



Rys. 1-1 **Przykład zakleszczenia procesów P1 i P2**

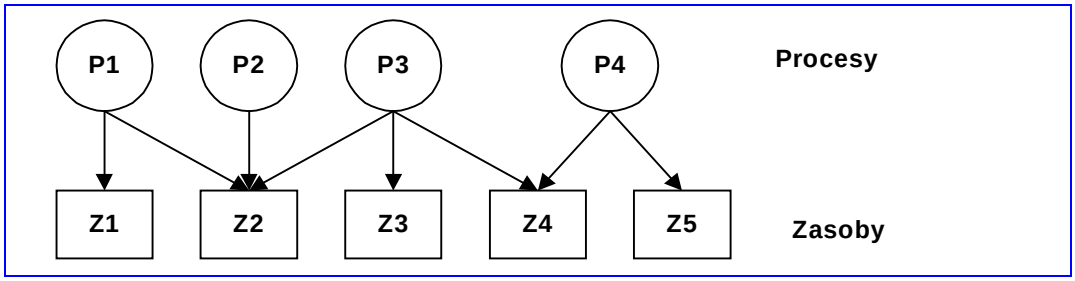


Rys. 1-2 Sieć Petriego ilustrująca zakleszczenie - zastój meksykański

Model systemu

System składa się z :

1. Skończonej liczby zasobów Z = **{Z_1, Z_2,...Z_n},** każdy z zasobów może wystąpić w pewnej liczbie egzemplarzy
2. Pewnej liczby procesów P = {**P_1, P_2,...,P_m**} które do wykonania swych zadań potrzebują zasobów Z.



Rys. 1-3 Model zasobów i procesów

Proces korzysta z zasobów w następujący sposób:
1. Zamawia dany zasób. Gdy zasób nie jest dostępny proces może być zmuszony do oczekiwania na zasób.
2. Używa danego zasobu. W tym czasie zasób nie jest dostępny dla innych procesów.
3. Zwalnia zasób.

Zamówienie i zwolnienie zasobu odbywa się to poprzez wykonanie pewnej funkcji systemowej (np. semaforowej).

Definicja zakleszczenia

Zbiór procesów jest w stanie zakleszczenia jeżeli każdy proces z tego zbioru czeka na zdarzenie które może być spowodowane tylko przez inny proces z tego samego zbioru.

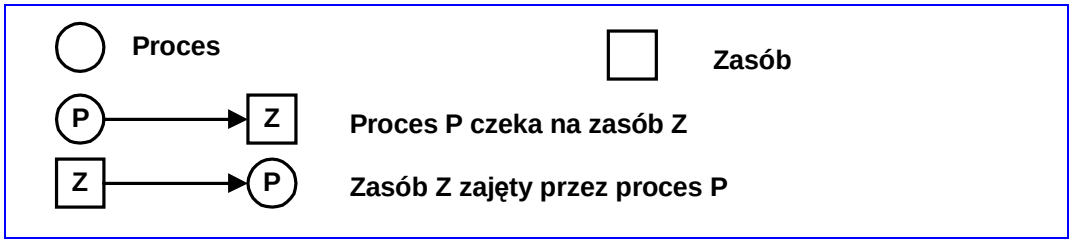
Do analizy systemów mogących wejść w stan zakleszczenia stosuje się grafy przydziału zasobów.

W grafie tym występują dwa rodzaje wierzchołków.

1. Pierwszy rodzaj odpowiada procesom P reprezentowanym jako kółka.
2. Drugi rodzaj wierzchołków odpowiada zasobom Z reprezentowanym jako kwadraty.

Łuki grafu reprezentują zależności pomiędzy procesami i zasobami.

1. Łuk biegnący od procesu P do zasobu Z oznacza że proces P jest zablokowany w oczekiwaniu na zasób Z.
2. Łuk biegnący od zasobu Z do procesu P oznacza że zasób Z jest zajęty przez proces P.



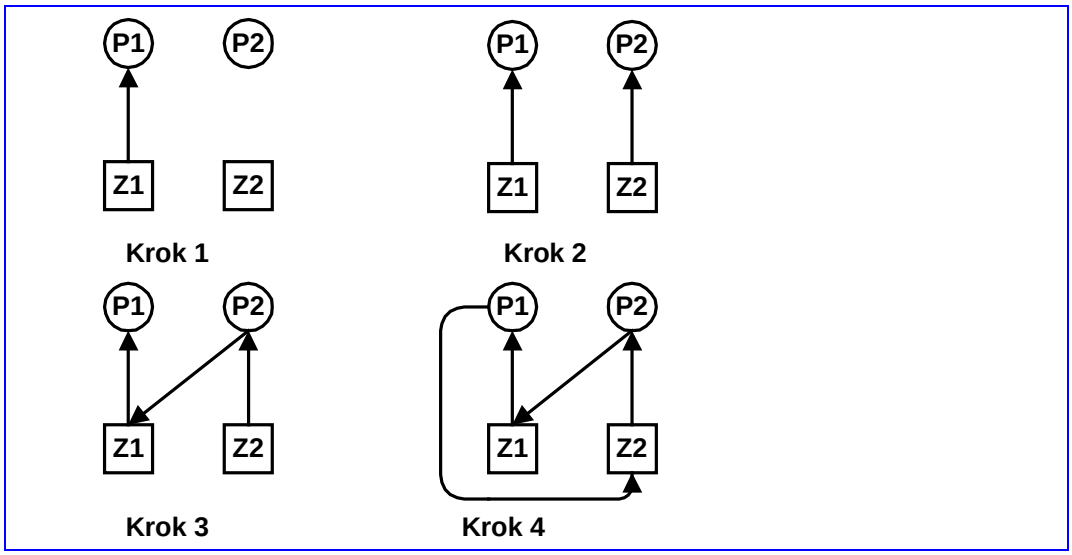
Rys. 1-4 **Węzły i łuki w grafie przydziału zasobów.**

Gdy proces oczekiwał na zasób i zasób ten został przydzielony to kierunek łuku (od procesu do zasobu) się zmienia na przeciwny (od zasobu do procesu).

Analiza zastoju meksykańskiego przy wykorzystaniu grafu przydziału zasobów.

Krok 1	Proces P1 zajmuje zasób Z1. Sterowanie przekazane zostaje do procesu P1.
Krok 2	Proces P2 zajmuje zasób Z2.
Krok 3	Proces P2 żąda zasobu Z1. Zasób jest zajęty więc proces P2 blokuje się w oczekiwaniu na zasób Z1.
Krok 4	Proces P1 żąda zasobu Z2. Zasób jest zajęty więc proces P1 blokuje się w oczekiwaniu na zasób Z2. Procesy P1 i P2 wchodzą w stan zakleszczenia.

Tab. 1 Kroki działania procesów P1 i P2 prowadzące do zakleszczenia.



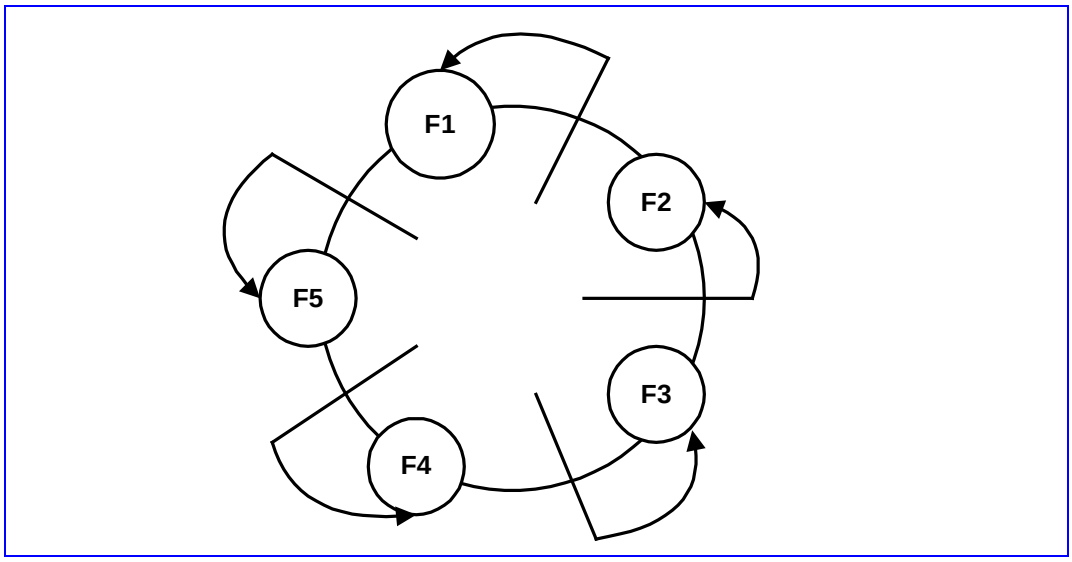
Rys. 1-5 **Ilustracja etapów zakleszczenia przy pomocy grafu przydziału zasobów**

Problem pięciu ucztujących filozofów

Przy okrągłym stole siedzi pięciu filozofów. Zajmują się oni naprzemiennie tylko dwoma czynnościami - myśleniem i jedzeniem. Do jedzenia filozof potrzebuje dwu widelców. Gdy filozof otrzyma dwa widelce je, a następnie odkłada obydwa widelce. Problem polega na takim zorganizowaniu pracy filozofów aby spełnione były warunki:

- Filozof je wtedy gdy zdobędzie dwa widelce.
- Dwu filozofów nie może trzymać tego samego widelca.
- Każdy z filozofów musi się w końcu najeść (nie zostanie zagłodzony).

Dodatkowym wymaganiem jest efektywność rozwiązania. Znaczy to że nie należy blokować aktywności filozofa gdy nie jest to konieczne dla spełnienia poprzednich warunków.

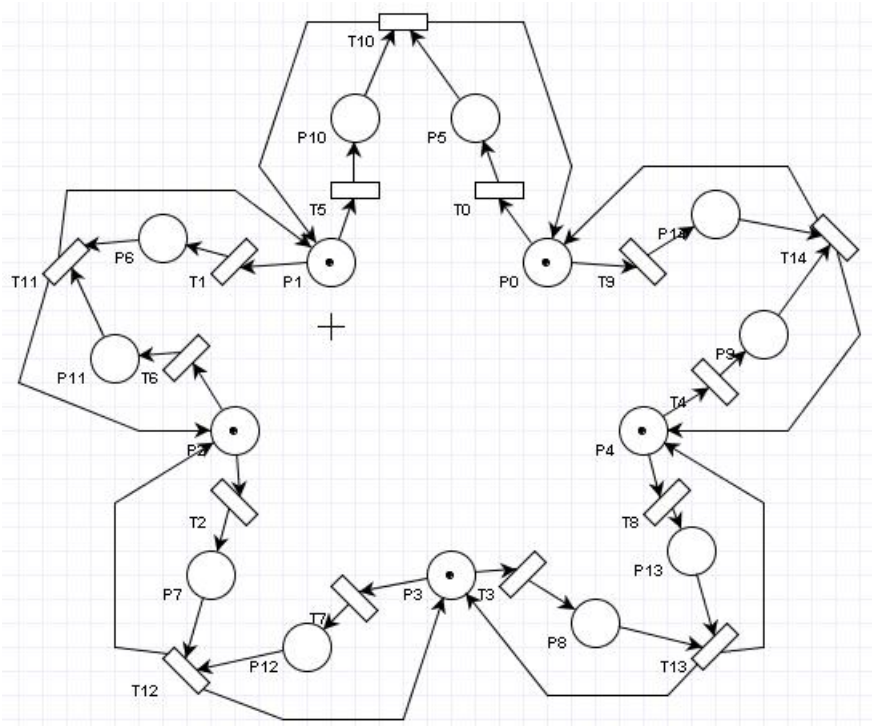


Rys. 1-6 **Ilustracja problemu pięciu ucztujących filozofów**

A. Rozwiązanie z możliwością blokady:

1. Filozof czeka aż będzie wolny lewy widelec i podnosi go.
2. Filozof czeka aż będzie wolny prawy widelec i podnosi go.
3. Filozof je.
4. Filozof odkłada obydwa widelce.
5. Filozof myśli.

Jeżeli w pewnej chwili każdy z filozofów podniesie lewy widelec i będzie czekał na prawy nigdy go nie otrzyma gdyż algorytm przewiduje zwolnienie widelców po zakończeniu jedzenia. Nastąpi zakleszczenie.



Rys. 1-7 Sieć Petriego dla problemu pięciu filozofów – filozof może podnieść prawy lub lewy widelec

Petri net state space analysis results

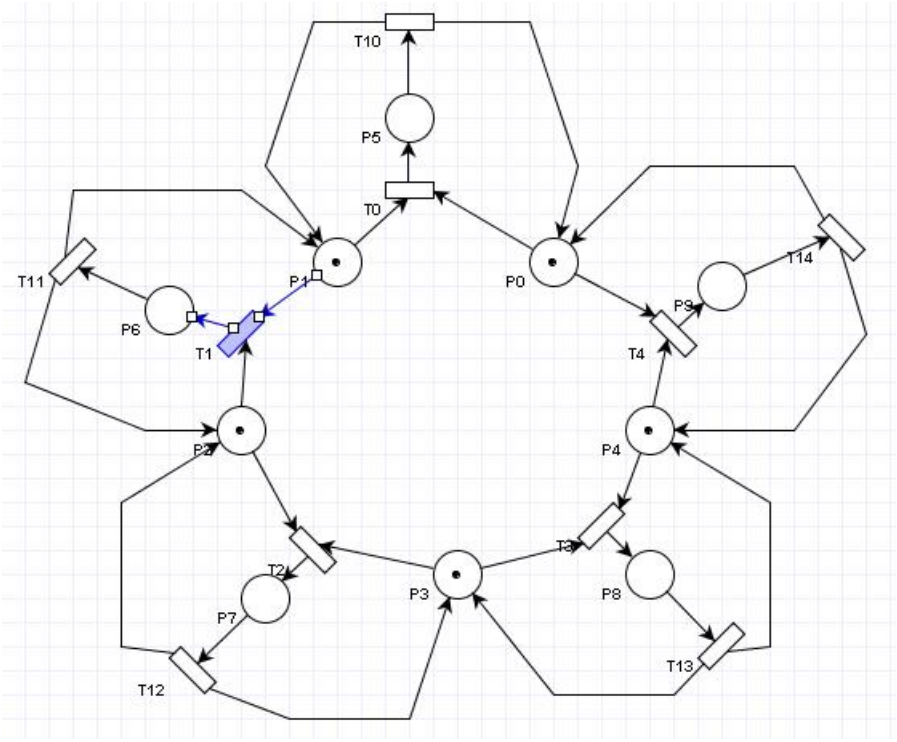
Bounded	true
Safe	true
Deadlock	true

Wynik 1-1 Wynik analizy sieci Petriego– filozof może podnieść prawy lub lewy widelec

B. Rozwiązanie z możliwością zagłodzenia:

1. Filozof czeka aż będą wolne oba widelce i podnosi je (musi to być operacja niepodzielna).
2. Filozof je.
3. Filozof odkłada obydwa widelce.
4. Filozof myśli.

Może się tak zdarzyć że filozof będzie miał bardzo żarłocznych sąsiadów z których w każdej chwili jeden z nich je. W takim przypadku filozof zostanie zagłodzony.



Rys. 1-8 Sieć Petriego dla problemu pięciu filozofów – filozof może podnieść tylko obydwa widelce naraz

Petri net state space analysis results

Bounded	true
Safe	true
Deadlock	false

Wynik 1-2 Wynik analizy sieci Petriego – filozof może podnieść tylko obydwa widelce naraz

Zakleszczenia

C. Rozwiązanie poprawne

1. Potrzebny jest arbiter zewnętrzny (nazywany lokajem) który dba o to aby jednej chwili najwyżej czterech filozofów konkurowało o widelce.
2. Dalej postępujemy jak w przypadku A.

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#define N 10 /* Liczba prob      */
#define P 5  /* Liczba  filozofow*/

typedef   enum { False=0, True=1 } bool ;

sem_t lokaj;
sem_t widelec[P];
bool Wybor ;

void *filozof(void *ptr) {
    int i, k = *((int *) ptr);
    for(i = 1; i <= N; i++) {
        printf("%*cMysli %d %d\n", k*4, ' ', k, i);
        if(Wybor) {
            sem_wait(&lokaj) ;
        }
        sem_wait(&widelec[k]) ;
        sem_wait(&widelec[(k+1) % P]) ;
        printf("%*cEat %d %d\n", k*4, ' ', k, i);
        sem_post(&widelec[k]) ;
        sem_post(&widelec[(k+1) % P]) ;
        if(Wybor) {
            sem_post(&lokaj) ;
        }
    }
    pthread_exit(0);
}
```

```
int main(int argc, char * argv[]) {
    int i, targ[P];
    pthread_t thread[P];
    sem_init(&lokaj, 0, P-1);
    Wybor = 1; /* Lub 0 gdy nie ma lokaja */
    printf("Wybor=%s\n",(Wybor?"true":"false"));
    for(i=0;i<P;i++) {
        sem_init(&widelec[i], 0, 1);
    }
    for(i=0;i<P;i++) {
        targ[i] = i;
        pthread_create(&thread[i], NULL, &filozof,
                       (void *) &targ[i]);
    }
    for(i=0;i<P;i++) {
        pthread_join(thread[i], NULL);
    }
    for(i=0;i<P;i++) {
        sem_destroy(&widelec[i]);
    }
    sem_destroy(&lokaj);
    return 0;
}
```

Przykład 1-2 Rozwiązanie problemu pięciu filozofów za pomocą semaforów

D. Rozwiązanie z monitorem (Ben Ari)

```
MONITOR zastawa {
int widelec[5];  // Liczba widelców dostępna dla fil. i
CONDITION warunek[5];  // Zmienna warunkowa

 void wez(int i) {
  if (widelec[i] != 2) Wait(warunek[i]);
  widelec[(i+1) %5]--;
  widelec[(i-1) %5]--;
 }

 void zwroc(int i) {
  widelec[(i+1) %5]++;
  widelec[(i-1) %5]++;
  if(widelec[(i+1)%5] == 2) Signal(warunek[(i+1%5]);
  if(widelec[(i-1)%5] == 2) Signal(warunek[(i-1%5]);
 }
} // monitor

void filozof(int i) {
 while(1) {
   // Myslenie

   wez(i);
   // Jedzenie
   zwroc(i);
 }
}

main {
  for(i=0;i<5;i++) widelec[i] = 2;
  ....
}
```

Przykład 1-3 Rozwiązanie problemu pięciu filozofów za pomocą monitora

Warunki Cofmanna

Warunki konieczne zakleszczenia podane zostały przez Coffmana.

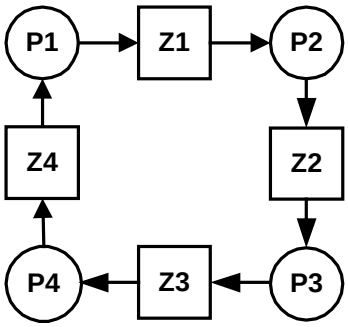
Do zakleszczenia może dochodzić gdy spełnione są jednocześnie cztery warunki:

1. Wzajemne wykluczanie (*ang. Mutual exclusion condition*) – Każdy z zasobów jest albo wolny albo zajęty przez dokładnie jeden proces.

2. Przetrzymywanie i oczekiwanie (*ang. Wait and hold condition*) – Proces posiadający jakiś zasób może żądać innego zasobu.

3. Brak wywłaszczeń (*ang. No prememption condition)* – Zasoby nie podlegają wywłaszczeniu, czyli zasób zajęty przez jakiś proces może być zwolniony tylko przez ten proces.

4. Czekanie cykliczne (*ang. Circular wait condition*) – Musi istnieć zamknięty łańcuch procesów **P = {P_1, P_2,...,P_m}** w którym **Pi** czeka na zasób zajęty przez **Pi+1** dla **i=1,m-1** oraz **Pm** czeka na na zasób zajęty przez **P1**.



Rys. 1-9 Graf przydziału zasobów dla przypadku czekania cyklicznego

Metody postępowania z zakleszczeniami

1. Zignorowanie problemu - nie zajmować się problemami zakleszczeń na poziomie systemu operacyjnego
2. Zapobieganie zakleszczeniom (*ang. deadlock prevention*) - stosować taki protokół zamawiania zasobów aby któryś z warunków Coffmana nie był spełniony i tym samym nie doszło do zakleszczenia.
3. Unikanie zakleszczeń *(ang. deadlock avoidance)* - stosować taki protokół zamawiania zasobów aby liczba liczba wolnych zasobów zawsze umożliwiała zakończenie rozpoczętych zadań.
4. Dopuszczać do możliwości zakleszczenia ale wykrywać je a następnie usuwać.

Zapobieganie zakleszczeniom

Zapobieganie zakleszczeniom (*ang. deadlock prevention*) opiera się na niedopuszczeniu do spełnienia jednego z warunków Coffmana.

Negacja warunku wzajemnego wykluczania

Nie jest możliwe uniknięcie tego warunku gdyż pewne zasoby są z natury niepodzielne.

Negacja warunku przetrzymywania i oczekiwania

Podejście polega na unikaniu sytuacji gdy proces posiadający już zasoby zamawia inne zasoby.

1. Proces może zająć zasoby tylko wtedy gdy wszystkie z nich może otrzymać.
2. Proces może zamówić jakieś zasoby tylko wtedy gdy nie posiada zajętych innych zasobów . Strategia nazywa się wszystko albo nic (*ang. One-shot allocation*).

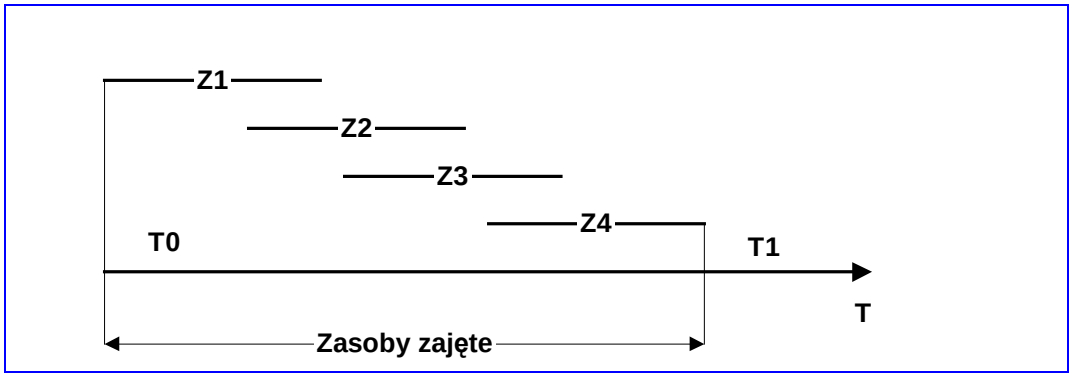
Zastosowanie strategii wszystko albo nic do problemu 5-ciu filozofów:

- Filozof może otrzymać albo dwa widelce albo żadnego.
- Filozof 1 i 3 otrzymują widelce – reszta czeka.

Rozwiązanie nie powoduje zakleszczenia ale może prowadzić do zagłodzenia.

Wady:

1. Małe wykorzystanie zasobów gdyż być może nie wszystkie naraz są potrzebne do wykonania zadania.
2. Możliwość zagłodzenia pewnych procesów poprzez ciągłe zajęcie często używanych zasobów.



Rys. 1-10 Zasoby Z1, Z2, Z3, Z4 zajęte w czasie [T0, T1].

Zasoby Z1, Z2, Z3, Z4 są zajęte w całym czasie [T0, T1] co nie jest koniecznie potrzebne.

Negacja warunku braku wywłaszczeń.

Możliwe jest zastosowanie następujących algorytmów:

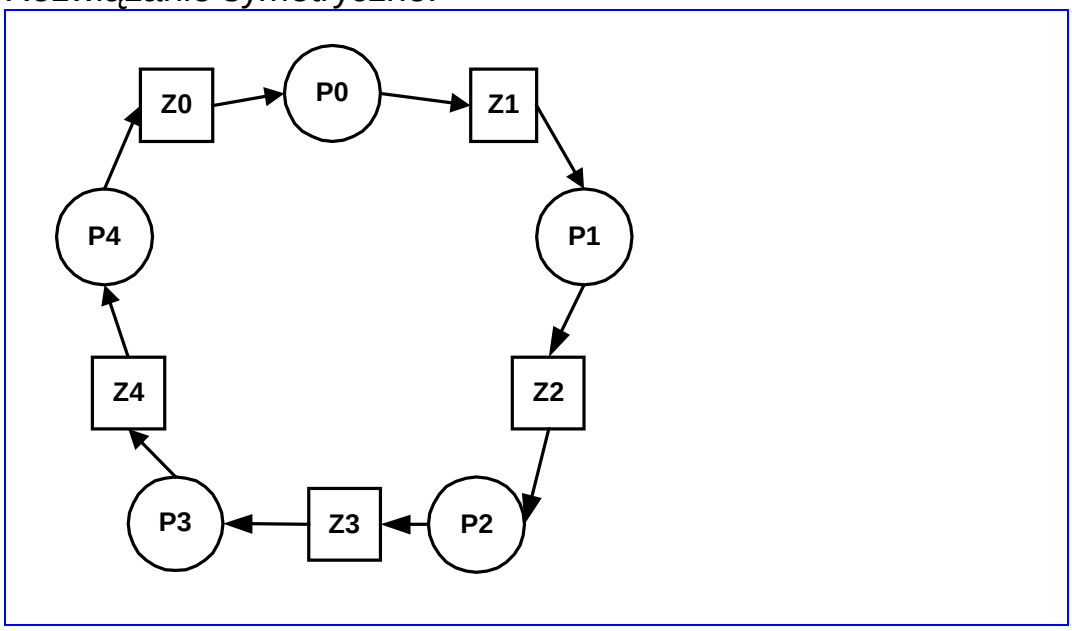
1) Gdy proces żąda zasobu który nie jest dostępny sprawdza się czy proces przytrzymujący te zasoby czeka na inne zasoby. Gdy tak jest odbiera się mu te zasoby. Implementacja takiej metody wymaga aby program był napisany w języku obsługującym wyjątki.
2) Gdy proces posiadający jakieś zasoby żąda innych zasobów które nie mogą być przydzielone – traci przydzielone zasoby.

Negacja warunku czekania cyklicznego

Aby zapobiec czekaniu cyklicznemu należy ponumerować zasoby i żądać aby każdy proces zamawiał zasoby we wzrastającym porządku numeracji. Gdy tak będzie nie wystąpi warunek czekania cyklicznego.

Negacja warunku czekania cyklicznego w problemie pięciu filozofów.

Rozwiązanie symetryczne:



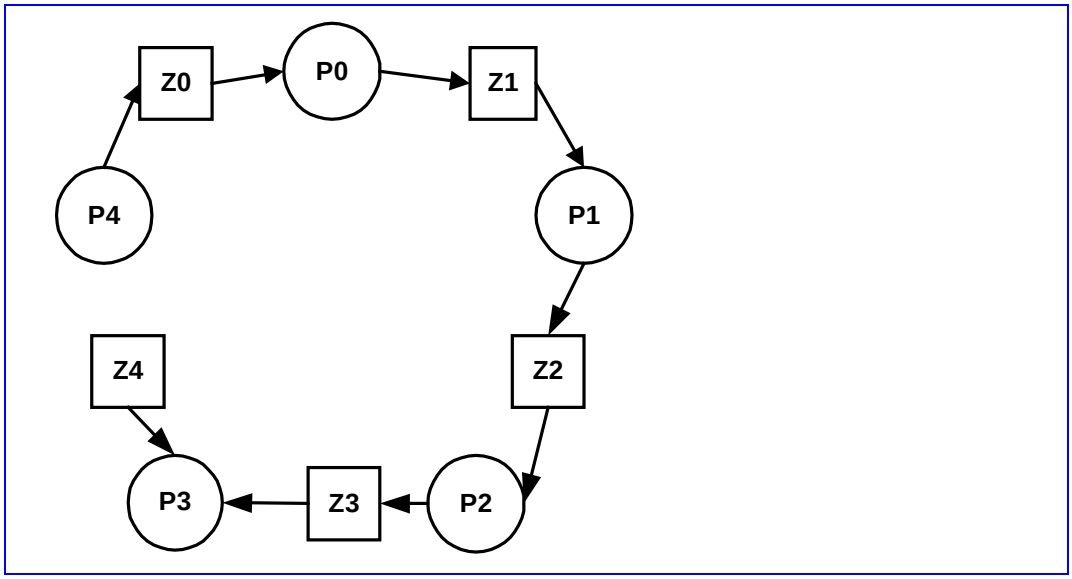
Rys. 1-11 Graf przydziału zasobów dla problemu pięciu filozofów – rozwiązanie symetryczne

Filozof	Pierwsze żądanie	Drugie żądanie
0	0	1
1	1	2
2	2	3
3	3	4
4	4	0

Przykład 1-4 Kolejność żądań zasobów w rozwiązaniu symetrycznym

W powyższym przypadku istnieje zamknięty cykl procesów.

Rozwiązanie asymetryczne:



Rys. 1-12 Graf przydziału zasobów dla problemu pięciu filozofów – rozwiązanie asymetryczne

Filozof	Pierwsze żądanie	Drugie żądanie
0	0	1
1	1	2
2	2	3
3	3	4
4	0	4

Przykład 1-5 Kolejność żądań zasobów w rozwiązaniu niesymetrycznym

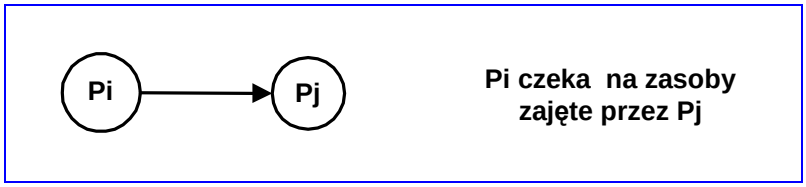
W kroku 1 zasób Z0 żądany przez P0 i P4. Obojętnie komu zostanie przydzielony w kroku 2 jeden z procesów otrzyma wszystkie konieczne zasoby.

Wykrywanie zakleszczeń

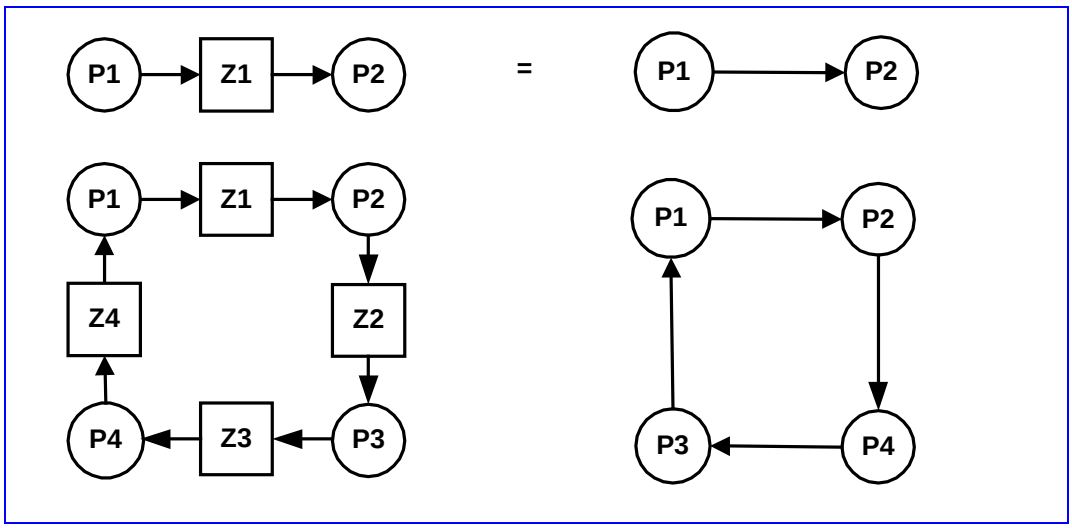
Założenie: zasoby reprezentowane jednokrotnie

Zakleszczenia mogą być wykryte na podstawie analizy grafu oczekiwania (*ang. wait-for graph*). Graf oczekiwania można uzyskać z grafu przydziału zasobów przez usunięcie wierzchołków odpowiadających zasobom.

Graf oczekiwania:
1. Wierzchołki – procesy {P1,P2,...Pn}
2. Luki - od Pi do Pj wtedy proces Pi czeka na zasoby zajęte przez proces Pj.



Rys. 1-13 Łuk grafu oczekiwania



Rys. 1-14 Otrzymywanie grafu oczekiwania z grafu przydziału zasobów

Twierdzenie

Gdy w grafie oczekiwania wystąpi cykl to system jest w stanie zakleszczenia.

Aby na bieżąco wykrywać zakleszczenia system musi:

1. Utrzymywać aktualny graf oczekiwania
2. Dla grafu oczekiwania wykonywać algorytm wykrywania cyklu

Gdy n – liczba wierzchołków to złożoność algorytmu n^2.

Likwidowanie zakleszczenia

Gdy zakleszczenie wystąpi i zostanie wykryte należy je zlikwidować. Możliwe podejścia:

Zakończenie procesów

- Wszystkich
- Niektórych aby zakleszczenie ustąpiło

Występuje problem które procesy zakończyć. Należy minimalizować koszty zakończenia przedwczesnego procesów. Decyzja zależy od:

- bezpieczeństwo
- priorytet procesu
- czas wykonywania procesu
- łatwość wywłaszczenie z zasobu

Wywłaszczenia z zasobów

Przy wywłaszczaniu zasobów należy rozważyć następujące kwestie

- Który z zasobów ma być wywłaszczony
- Jak przeprowadzić wycofanie
- Jak zapewnić aby nie doszło do zagłodzenia procesu