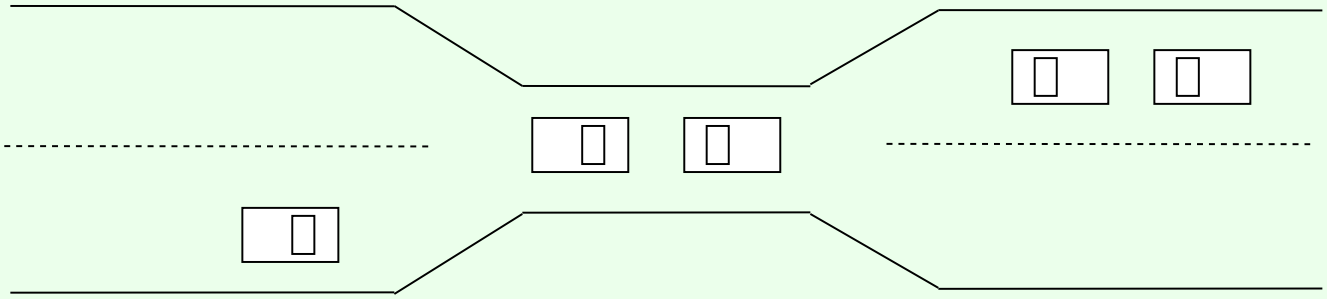
ZAKLESZCZENIA



- w SO brak środków zapobiegania zakleszczeniom
- Zamówienia na zasoby =>

  przydział dowolnego egzemplarza danego typu
- Zasoby w systemie – typy; identyczne egzemplarze
  - procesory
  - obszary pamięci
  - cykle procesora
  - pliki
  - urządzenia we/wy
- Porządek dostępu do zasobu:
  - zamówienie (funkcje systemowe, operacje semaforowe)
  - użycie
  - zwolnienie

- Przed użyciem zasobu SO sprawdza w tablicy systemowej, czy zasób został przydzielony procesowi
- Tablica systemowa:
  - zasób
  - wolny/zajęty
  - id procesu, któremu przydzielono zasób
  - kolejka procesów oczekujących na zasób

Zbiór procesów jest w stanie **zakleszczenia**

<u>każdy</u> proces z tego zbioru czeka na zdarzenie, które może być efektem działania innego procesu z tego zbioru

zdarzenia – przydział, zwalnianie zasobów fizycznych, logicznych

- komunikacja między procesami

Warunki konieczne do wystąpienia zakleszczenia

- Wzajemne wykluczanie

przynajmniej jeden zasób – niepodzielny

- Przetrzymywanie i oczekiwanie

  Jeden proces dysponuje jednym zasobem, oczekuje na drugi zasób, który jest przetrzymywany przez inny proces

- Brak wywłaszczeń

  Zasób może być zwolniony tylko przez proces, któremu został przydzielony

- Czekanie cykliczne

  procesy czekające $\{P_0, P_1, P_2, \ldots P_n\}$ tworzą cykl:

$$P_0 \rightarrow P_1 \rightarrow P_2 \rightarrow \ldots P_n \rightarrow P_0$$

Graf przydziału zasobów

Resource-Allocation Graph

zbiór wierzchołków - *V*

zbiór krawędzi - *E*

- V:

  $P = \{P_1, P_2, \ldots, P_n\}$ - procesy

  $R = \{R_1, R_2, \ldots, R_m\}$ – zasoby

- $P_i \rightarrow R_j$

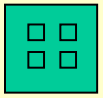
  Krawędź zamówienia – krawędź skierowana

- $R_j \rightarrow P_i$

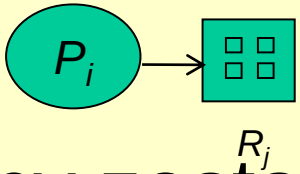
  Krawędź przydziału

- proces

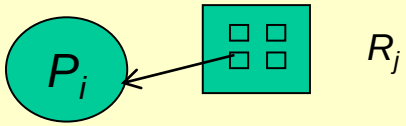
- 4 egzemplarze jednego typu zasobu



- P_i żąda zasobu R_j



- R_j przydzielony zostaje P_i

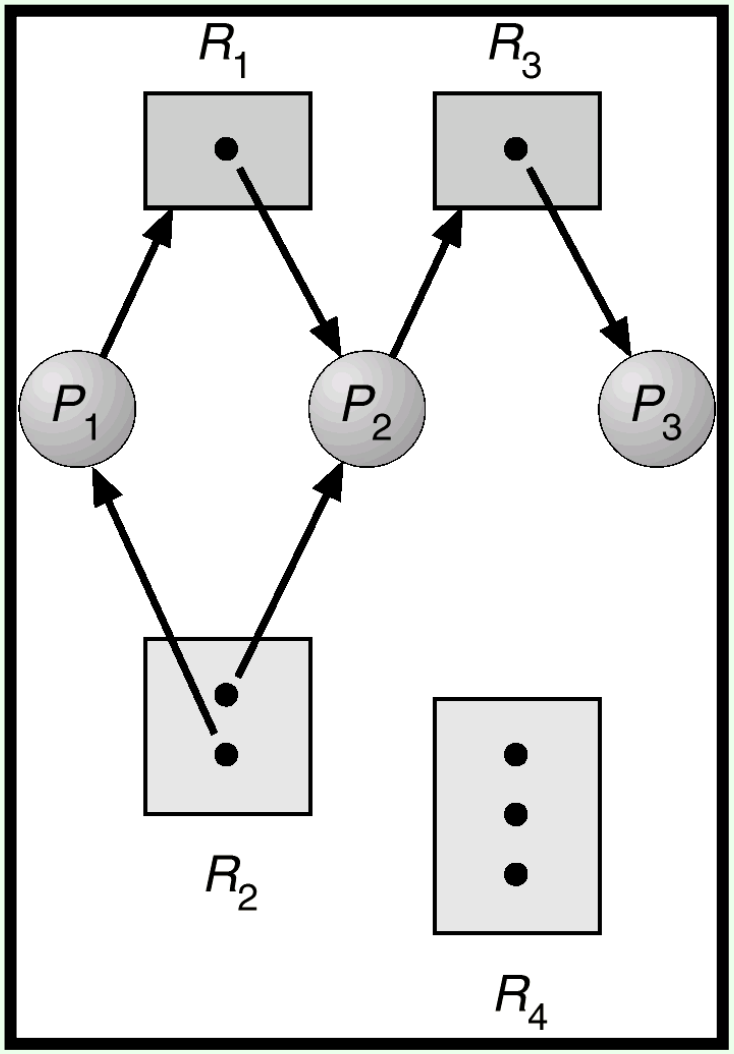


- Przydział zasobu: - krawędź zamówienia zastąpiona krawędzią przydziału

Brak cykli,

nie ma

zakleszczenia

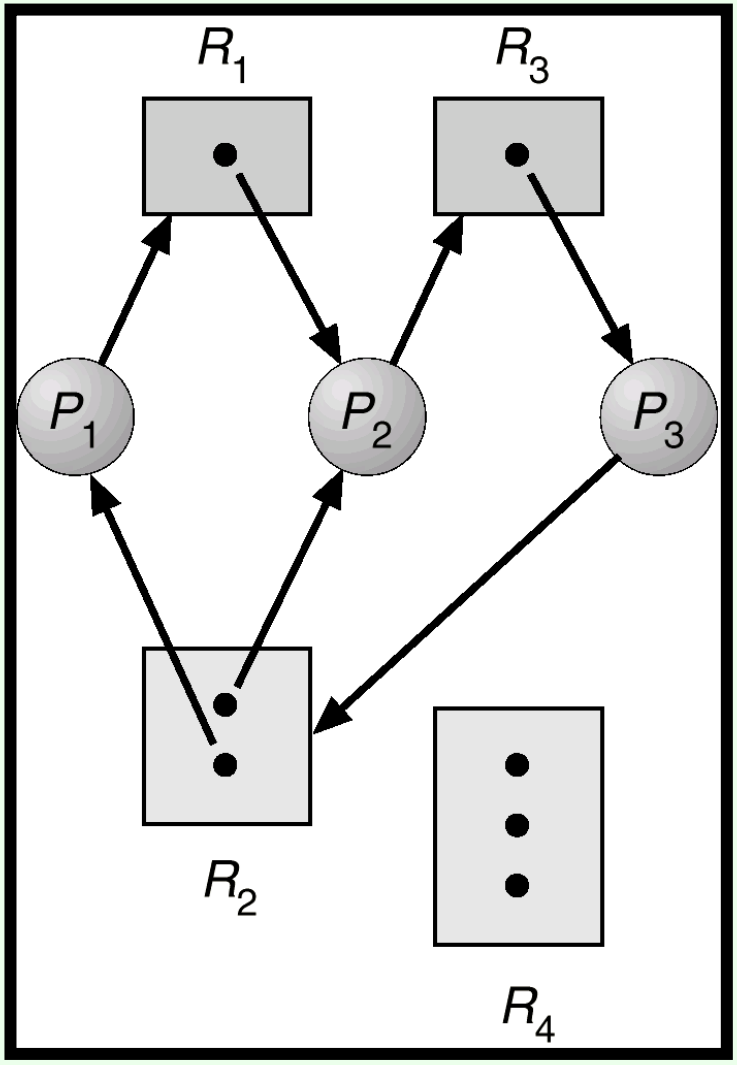


zakleszczenie

Cykle:

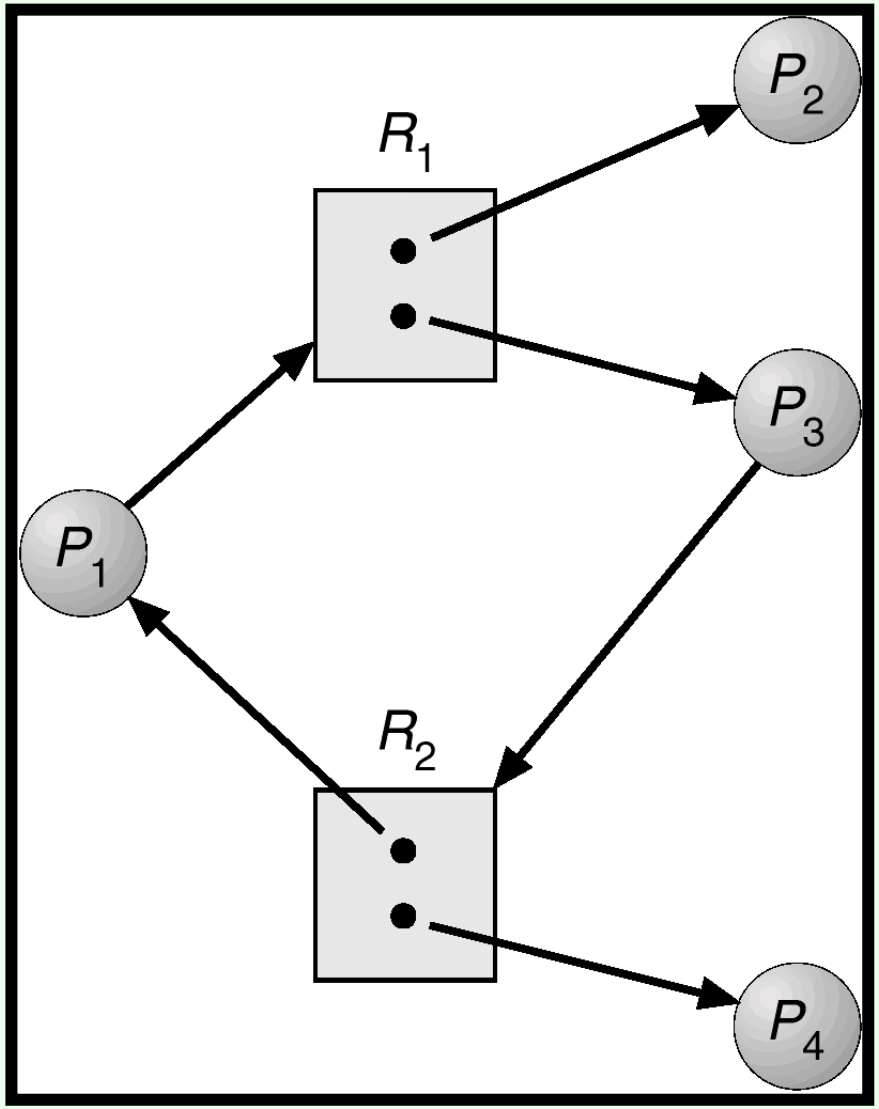
$P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$

$P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$



Jest cykl; bez zakleszczenia

Cykl:
$P_1 \rightarrow R_1 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$



- Nie ma cykli ⇒ nie ma zakleszczenia
- Są cykle ⇒
  - Jeśli po jednym egzemplarzu każdego zasobu - zakleszczenie
  - jeśli po kilka egzemplarzy każdego typu zasobu – możliwe zakleszczenie

Co robić z zakleszczeniami

- Gwarancja, że system nigdy nie wejdzie w stan zakleszczenia
- Dopuszczanie wystąpienia zakleszczeń + usuwanie zakleszczeń
- Ignorowanie problemu; założenie, że zakleszczenie nigdy nie wystąpi - UNIX

Zapobieganie zakleszczeniom (1)

Nie spełnienie min. jednego z 4 warunków

- **Wzajemne wykluczanie**

  niemożliwe dla zasobów niepodzielnych

- **Przetrzymywanie i oczekiwanie**

  potrzeba zagwarantowania, że gdy proces żąda zasobu nie dysponuje innym zasobem

–Alokowanie wszystkich zasobów procesu przed działaniem

–Zamawianie zasobu po zwolnieniu innych zasobów

Wady:

–Małe wykorzystanie zasobów

–Możliwe głodzenie

Zapobieganie zakleszczeniom (2)

- **Brak wywłaszczeń –**

–Protokół 1

- Jeśli zamówienie nie może być spełnione - utrata wszystkich zasobów
- Dodanie ich do listy wolnych zasobów
- Restart procesu możliwy gdy dostępne wszystkie wymagane zasoby

–Protokół 2

- Jeśli zamówione zasoby przydzielone są procesowi,
  który czeka na dodatkowy zasób – wywłaszczenie czekającego procesu i przydzielenie zasobów zamawiającemu

- Jeśli nie – proces zamawiający czeka;
- wywłaszczenie procesu zamawiającego - tylko wtedy,
  gdy inny proces zażąda jego zasobów

Zapobieganie zakleszczeniom (3)

- **Czekanie cykliczne**
  - –całkowite uporządkowanie typów zasobów
  - –zamawianie zasobów w kolejności

Unikanie zakleszczeń

Wymaga dodatkowych informacji *a priori* o zamawianych zasobach

- Każdy proces deklaruje max. liczbę zasobów każdego typu
- Dynamiczne sprawdzanie stanu alokacji zasobów, aby nie doszło do czekania cyklicznego
- Stan przydziału zasobów:

–liczba dostępnych i przydzielonych zasobów

–maksymalne zapotrzebowanie procesów

Stan bezpieczny

- System jest w stanie bezpiecznym – istnieje bezpieczny ciąg procesów

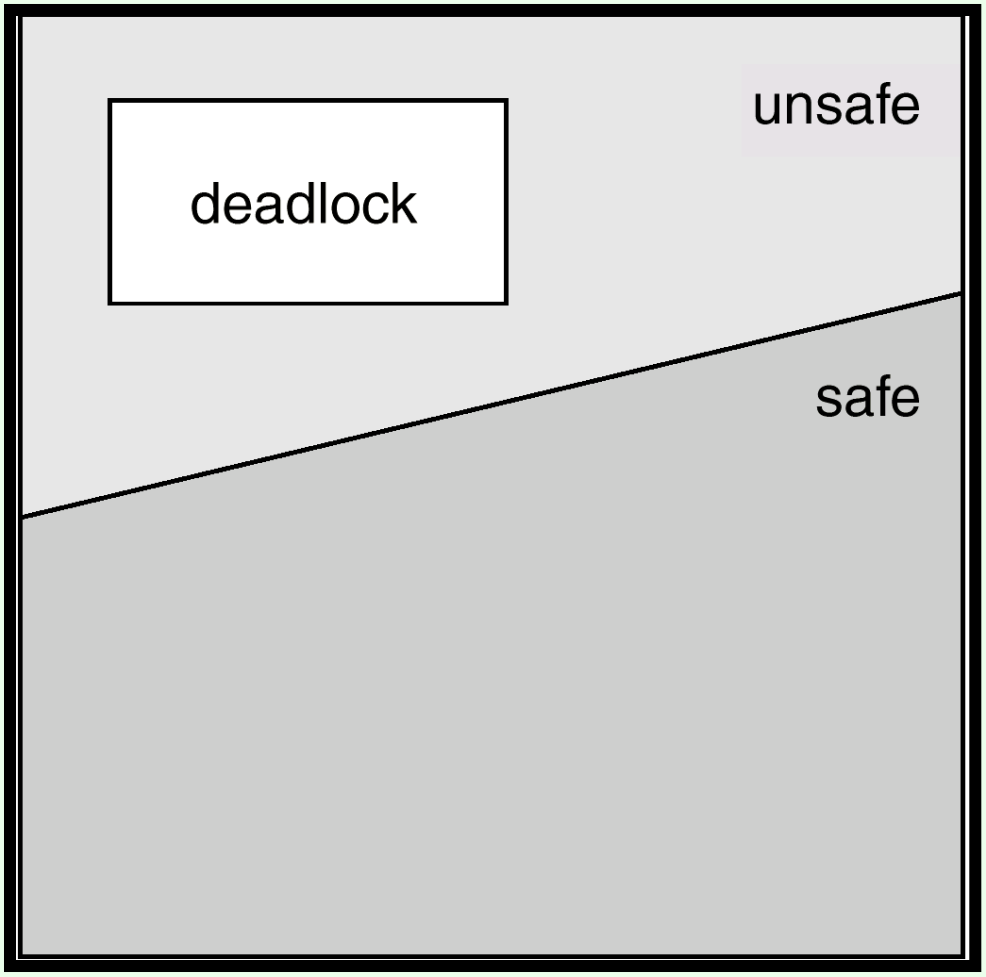
- Ciąg procesów <P_1, P_2, …, P_n> jest bezpieczny jeśli dla każdego P_i, jego potencjalne zapotrzebowanie na zasoby może być spełnione za pomocą aktualnie dostępnych zasobów + zasoby użytkowane przez inne P_k $k < i$

– If zasoby potrzebne P_i nie są dostępne natychmiast
  => P_i czeka, aż wszystkie P_k *się* zakończą
– Po zakończeniu P_k , P_i otrzymuje potrzebne zasoby, kończy działanie
– Po zakończeniu P_i , P_{i+1} i kolejne mogą otrzymać zasoby

- System w stanie bezpiecznym (*safe*) ➡
  nie ma deadlock'u

- System w stanie zagrożenia (*unsafe*) ➡
  możliwy deadlock

- Stan bezpieczny (brak zakleszczenia) ➡
  system nigdy nie wejdzie w stan zagrożenia


unsafe
deadlock
safe

przykład

Razem: 20

Proces	max.	aktualne	
P_0	8	5	
P_1	7	3	**5 /zakleszcz.**
P_2	11	9	

Stan bezpieczny systemu

$<P_o, P_1, P_2>$

$<P_o, P_2, P_1>$

$<P_2, P_1, P_0>$

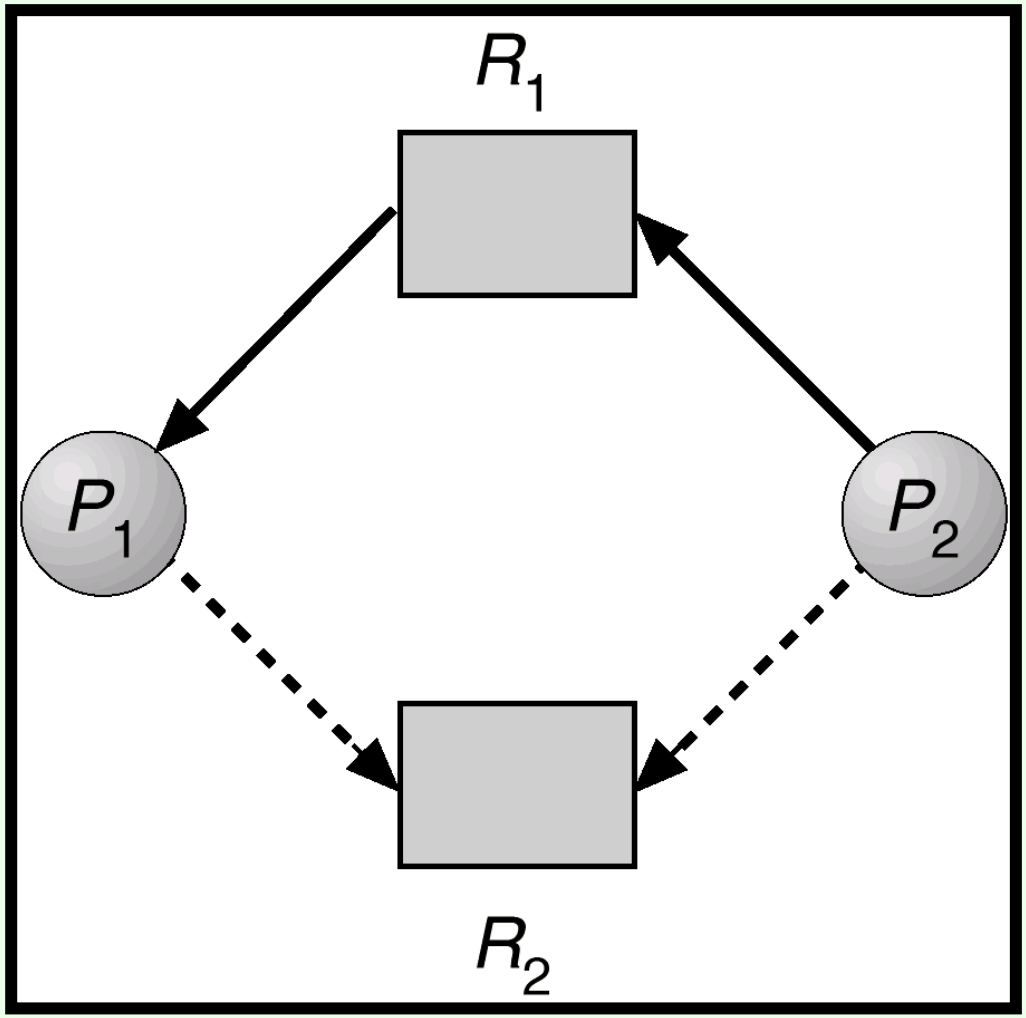
$<P_2, P_0, P_1>$

Algorytm grafu przydziału zasobów

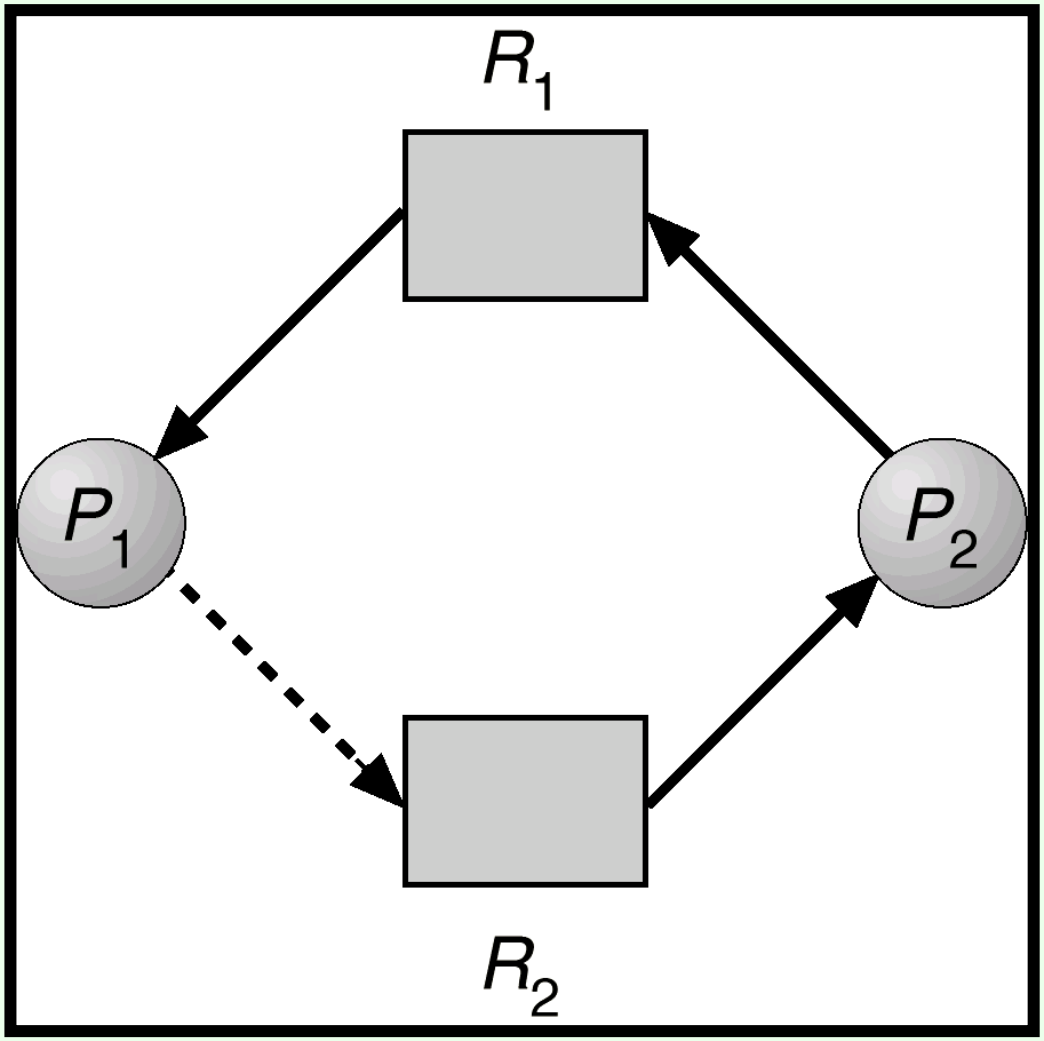
- Krawędź deklaracji $P_i \rightarrow R_j$ proces P_j może zażądać zasobu R_j; linia przerywana
- Krawędź deklaracji zamieniana na krawędź zamówienia gdy proces zażąda zasobu
- Po zwolnieniu zasobu przez proces krawędź zamówienia zamieniana na krawędź deklaracji
- Zasoby muszą być zamawiane *a priori* w systemie

Unikanie zakleszczeń; po 1 egzemplarzy zasobów

Graf przydziału zasobów – unikanie zakleszczeń



Graf przydziału zasobów – stan zagrożenia



Algorytm Bankiera

- Każdy typ zasobu ma wiele egzemplarzy
- Każdy proces musi a priori zadeklarować maksymalną liczbę egzemplarzy zasobów, które będą mu potrzebne
- Liczba ta nie może przekraczać ogólnej liczby zasobów w systemie
- Gdy proces żąda zasobu, którego przydział spowodowałby wejście systemu w stan zagrożenia – musi poczekać
- Proces musi zwrócić wszystkie zasoby w skończonym czasie

Struktury Danych dla Algorytmu Bankiera

n = liczba procesów, *m* = liczba typów zasobów.

- **Available[m]**

  Available[j] = k, jest dostępnych k egzemplarzy zasobu typu R_j

- **Max[n,m]**

Max [i,j] = k, proces P_i może zażądać max. *k* egzemplarzy zasobu typu R_j.

- **Allocation[n, m]**

  Allocation[*i,j*] = *k* , P_i ma aktualnie *k* egzemplarzy R_j

- **Need[n, m]**

Need[i,j] = k, P_i może potrzebować *k* dodatkowych egzemplarzy R_j aby zakończyć zadanie

Need [i,j] = Max[i,j] – Allocation [i,j]

Algorytm – czy system jest w stanie bezpiecznym

1. Work[m], Finish[n]

Work = Available

Finish [i] = false for i - 1,3, …, n.

2. znajdź i :

(a) Finish [i] = false

(b) $Need_i$ <= Work

Jeśli nie ma i go to step 4

3. Work = Work + $Allocation_i$

Finish[i] = true

go to step 2

4. If Finish [i] == true for all i, then

system jest w stanie bezpiecznym

n = liczba procesów

m = liczba typów zasobów

$m*n^2$

Algorytm zamawiania zasobów

Request = wektor zamówień dla procesu P_i.
$Request_i[j] = k$ - proces P_i potrzebuje k egzemplarzy zasobu typu R_j.

1.If $Request_i <= Need_i$ **then** go to step 2
else error /proces przekroczył deklarowane max/

2. If $Request_i <= Available$ **then** go to step 3
else P_i musi czekać /zasoby nie są dostępne/

3. Zmiany stanu:

$Available = Available - Request_i;$
$Allocation_i = Allocation_i + Request_i;$
$Need_i = Need_i – Request_{i;}$

- If stan bezpieczny -> zasoby przydzielone procesowi P_i.
- If stan zagrożenia -> P_i musi czekać; przywrócenie poprzedniego stanu przydziału zasobów

Przykład - algorytm bankiera

•5 procesów P_0 - P_4; 3 typy zasobów: *A, B, C*
A (10 egzemplarzy),
B (5 egzemplarzy)
C (7 egzemplarzy).

•T_0:

	Allocation	*Max*	*Available*
	A B C	***A B C***	***A B C***
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

Need = Max – Allocation

	A B C
P_0	7 4 3
P_1	1 2 2
P_2	6 0 0
P_3	0 1 1
P_4	4 3 1

Stan – bezpieczny:

$< P_1, P_3, P_4, P_2, P_0 >$

P_1 żąda (1,0,2)

•Sprawdzenie: Request <= Available

(1,0,2) <= (3,3,2) -> *true.*

Need P_1 – 1 2 2

	Allocation	*Need*	*Available*
	A B C	***A B C***	***A B C***
P_0	0 1 0	7 4 3	2 3 0
P_1	3 0 2	0 2 0	
P_2	3 0 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

•<P_1, P_3, P_4, P_0, P_2> - spełnia wymagania bezpieczeństwa

•żądanie (3,3,0) dlaP_4 ?, (2,0,0) dla P_1

•żądanie (0,2,0) dlaP_0?, (6,0,0) dla P_2

Wykrywanie Deadlock'u

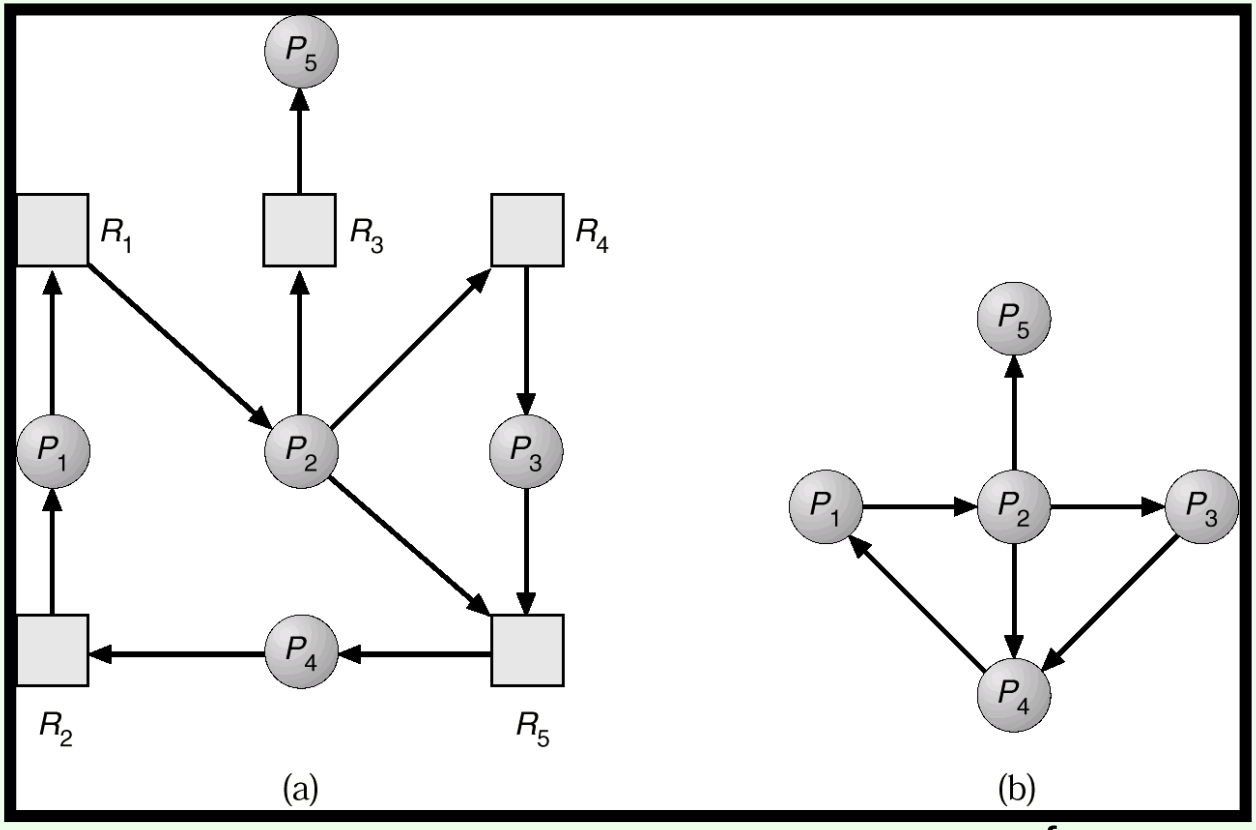
- Pozwolenie na wejście w stan zakleszczenia
  - Algorytm wykrywania zakleszczenia
  - Wyjście z zakleszczenia

Pojedyncze egzemplarze zasobów

- graf oczekiwania
  - Węzły - procesy
  - $P_i \rightarrow P_j$; P_i czeka na P_j.
- graf oczekiwania zawiera cykl => zakleszczenie
- algorytm wykrywania cykli w grafie - n^2

  (n – liczba wierzchołków w grafie)

graf oczekiwania



graf przydziału zasobów

graf oczekiwania

wielokrotne egzemplarze zasobów

- *Available[m]*

  liczba dostępnych zasobów każdego typu

- *Allocation[n, m]*

  liczba zasobów każdego typu aktualnie przydzielona każdemu z procesów

- *Request[n, m]*

  bieżące zamówienia każdego procesu

  Request [*i*,*j*] = *k*, proces P_i zamawia dodatkowo *k* egzemplarzy zasobu typu R_j

Algorytm wykrywania zakleszczeń

1. *Work[m], Finish[n]* :

   Work = Available

   for *i* = 1,2, …, *n*
   if $Allocation_i$!= 0 **then** *Finish*[i] = false
   else *Finish*[i] = *true*

2. znajdź *i* :

(a) *Finish*[*i*] == *false*
(b) $Request_i$ <= *Work*

   if nie ma takiego *i* **go to** 4.

3. *Work = Work + Allocation*$_i$

   Finish[*i*] = *true*
   go to 2

4. **if** *Finish*[*i*] == false dla *i:* 1 <= *i* <= *n* **then** system jest w stanie zakleszczenia

   if *Finish*[*i*] == *false* **then** P_i - zakleszczony

przykład

•5 procesów $P_{0,\ldots}$ P_4; 3 typy zasobów
A (7 egzemplarzy) *B* (2 egzemplarzy), *C* (6 egzemplarzy).

•T_0:

	Allocation	*Request*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2	
P_2	3 0 3	0 0 0	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

•$<P_0, P_2, P_3, P_1, P_4>$ => *Finish*[*i*] = true dla każdego *I*

P2 zamawia dodatkowo zasób C

Request

	A B C
P 0	0 0 0
*P*1	2 0 2
*P*2	0 0 1
*P*3	1 0 0
*P*4	0 0 2

Zakleszczenie procesów P_1, P_2, P_3, P_4

- Kiedy wywoływać algorytm wykrywania zakleszczeń
  - Jak często może wystąpić zakleszczenie?
  - Ilu procesów dotyczy?

- Gdy zamówienie nie może być spełnione natychmiast

- Gdy wykorzystanie procesora < 40%

- W grafie może powstać wiele cykli – problem z wykryciem „sprawcy”

usuwanie zakleszczeń kończenie procesów

- Usunięcie wszystkich zakleszczonych procesów
- Usuwanie procesów po kolei; powtarzanie algorytmu wykrywania zakleszczeń

- Kolejność wyboru procesów do usunięcia
  - Priorytet
  - Długość obliczeń i czas, który pozostał do zakończenia
  - Użytkowane zasoby procesu
  - Zasoby potrzebne do zakończenia
  - Liczba procesów do przerwania
  - Typ procesu (interakcyjny, wsadowy)

usuwanie zakleszczeń wywłaszczanie zasobów

- Wybór ofiary (zasób, proces)
- Co dalej z wywłaszczonym procesem?
- Niedopuszczanie do głodzenia

usuwanie zakleszczeń podejście mieszane

- 3 podstawowe podejścia
  - zapobieganie
  - unikanie
  - wykrywanie

- Podział zasobów na hierarchicznie uporządkowane klasy
- W obrębie klas – najodpowiedniejsze metody

usuwanie zakleszczeń podejście mieszane

- 4 klasy

–Zasoby wewnętrzne (PCB) – uporządkowanie zasobów
–Pamięć główna – wywłaszczenie
(obraz zadania do pamięci pomocniczej)
–Zasoby zadania – urządzenia – unikanie zakleszczeń
–Wymienny obszar pamięci pomocniczej – wstępny przydział