Zakleszczenia

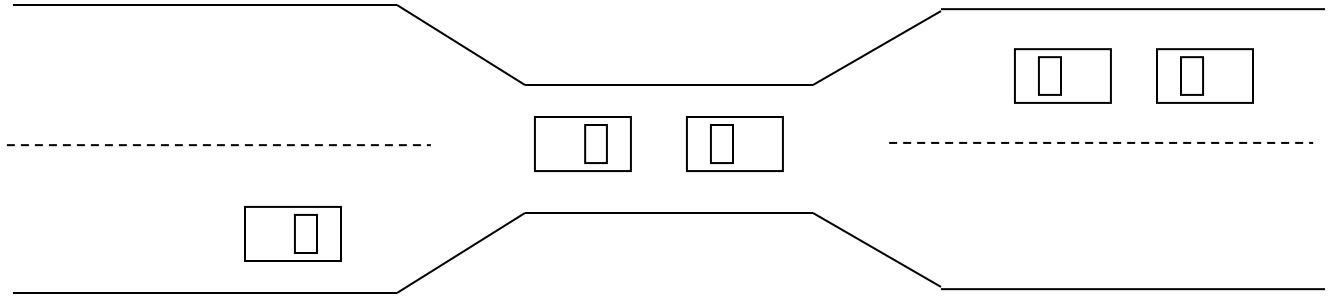
Ostatnia modyfikacja: 15.04.2020

Problem zakleszczenia

- Zakleszczenie polega na zablokowaniu zbioru procesów, z których każdy posiada pewien zasób i jednocześnie oczekuje na zasób będący w posiadaniu innego procesu tego zbioru.
- Przykład
  - System ma 2 przewijaki taśmy.
  - Procesy P_1 i P_2 zajęły po jednym przewijaku, ale potrzebują drugi.
- Przykład
  - semafory *A* i *B*, są inicjalizowane wartością 1

P_0	P_1
wait (A);	*wait(B)*
wait (B);	*wait(A)*

Przykład: przejazd przez wąski most



- Ruch jednokierunkowy.
- Każdy odcinek mostu można uważać za zasób.
- Jeśli wystąpi zakleszczenie, to należy wycofać co najmniej jeden pojazd (wywłaszczenie z zasobu i wycofanie), ale być może więcej pojazdów.
- Możliwe jest zjawisko głodzenia.

Model systemu

- Zasoby można poklasyfikować na typy R_1, R_2, . . ., R_m

  CPU , pamięć, urządzenia wejścia/wyjścia

- Każdy zasób typu R_i ma W_i egzemplarzy.
- Każdy proces korzysta z zasobu następująco:
  - zajęcie
  - wykorzystanie
  - zwolnienie

Charakterystyka zakleszczeń

Zakleszczenie ma miejsce przy jednoczesnym spełnieniu 4 warunków

- **Wzajemne wykluczanie:** tylko jeden proces ma w danym momencie dostęp do zasobu.
- **Przetrzymywanie i oczekiwanie:** proces zajmujący przynajmniej jeden zasób oczekuje na zasoby zajmowane przez inne procesy.
- **Brak wywłaszczania:** zasób może stać się wolny jedynie przez dobrowolne zwolnienie go przez proces, który go zajmuje.
- **Okrężne czekanie:** istnieje zbiór $\{P_0, P_1, \ldots, P_0\}$ oczekujących procesów taki, że P_0 oczekuje na zasób zajęty przez P_1, P_1 oczekuje na zasób zajęty przez P_2, …, P_{n-1} oczekuje na zasób zajęty przez P_n, a P_0 oczekuje na zasób zajęty przez P_0.

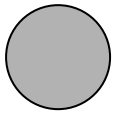
Graf alokacji zasobów

Graf składa się ze zbioru wierzchołków V i krawędzi E.

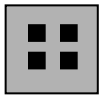
- V można podzielić na dwa podzbiory:
  - $P = \{P_1, P_2, \ldots, P_n\}$, zbiór wszystkich procesów w systemie.
  - $R = \{R_1, R_2, \ldots, R_m\}$, zbiór wszystkich typów zasobów w systemie.
- Krawędź zamówienia – krawędź skierowana $P_1 \rightarrow R_j$
- Krawędź przydziału – krawędź skierowana $R_j \rightarrow P_i$

Graf alokacji zasobów (c.d.)

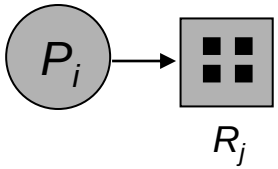
- Proces



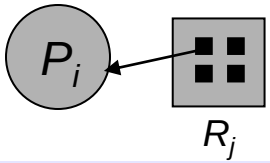
- Typ zasobu z 4 egzemplarzami



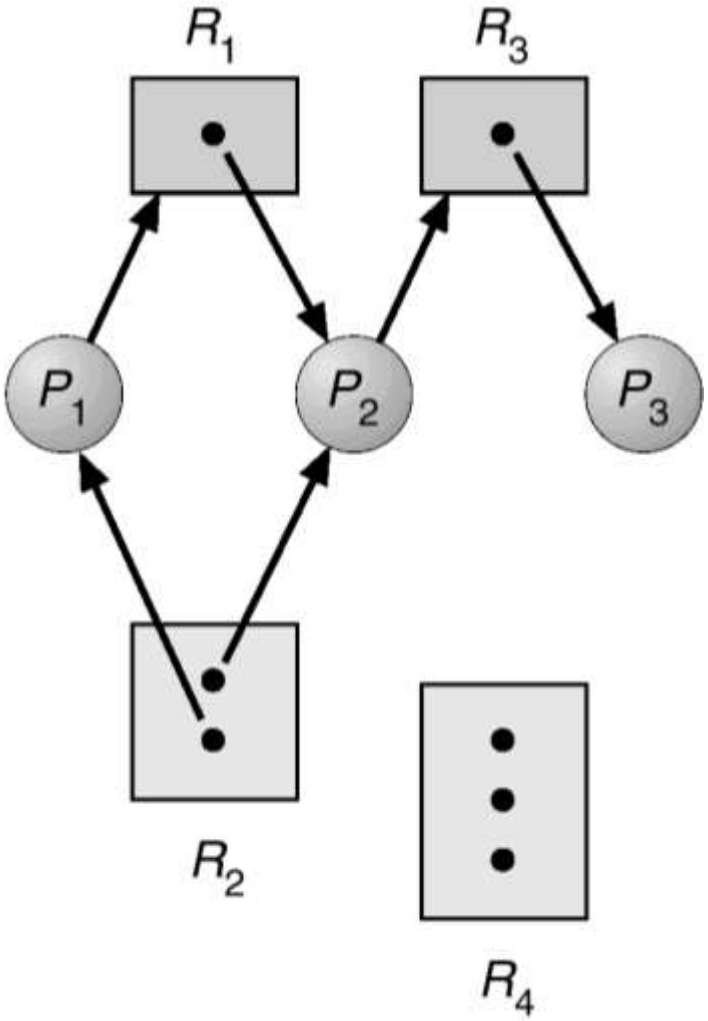
- P_i żąda egzemplarza typu R_j



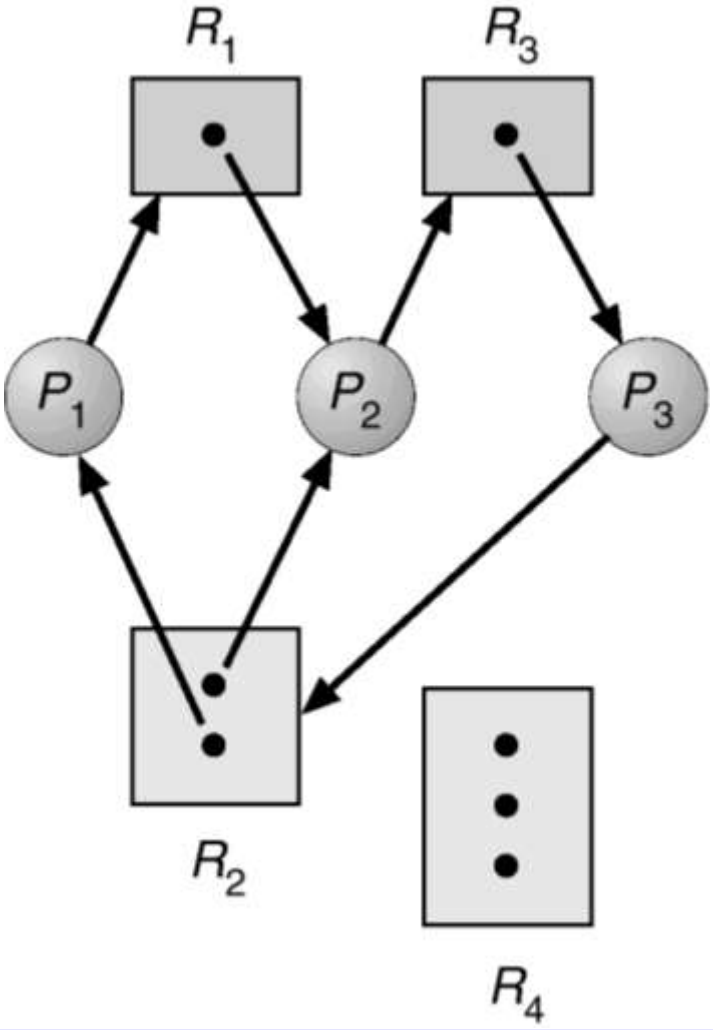
- P_i zajmuje egzemplarz typu R_j



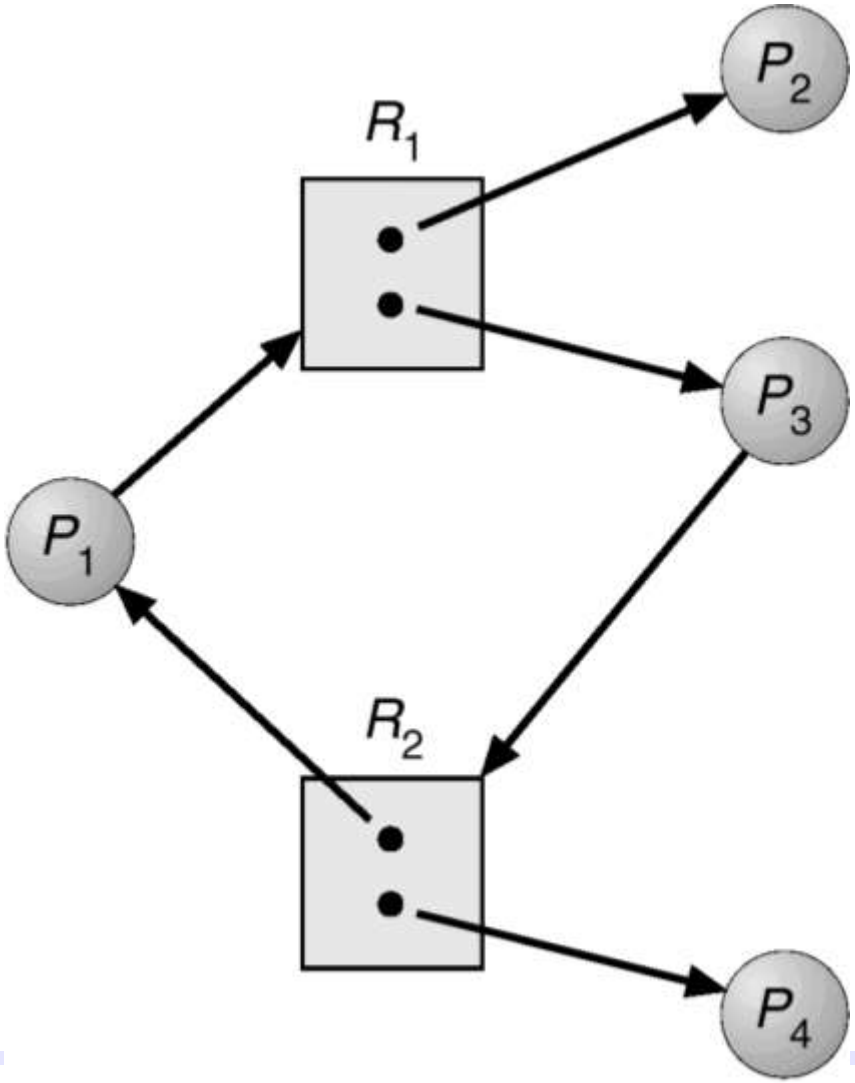
Przykład grafu alokacji zasobów



Graf alokacji zasobów z zakleszczeniem



Graf przydziału zasobów z cyklem, bez zakleszczenia



Podstawowe fakty

- Jeśli graf **nie zawiera cyklu** $\Rightarrow$ **brak zakleszczenia**.
- Jeśli graf **zawiera cykl** $\Rightarrow$
  - jeśli typy zasobów są **reprezentowane pojedynczo**, to **zakleszczenie**.
  - jeśli typy zasobów są reprezentowane przez **wiele egzemplarzy** - **zakleszczenie jest możliwe**.

Metody postępowania z zakleszczeniami

- Należy zapewnić, że system **nigdy** nie znajdzie się w stanie zakleszczenia.
- **Nie przeszkadzamy** systemowi znaleźć się w stanie zakleszczenia, ale umiemy **wydobywać system z tego stanu**.
- **Ignorujemy** problem, udając że problem zakleszczenia nigdy nie ma miejsca w naszym systemie (podejście to jest przyjmowane przez w większości systemów, w tym UNIX).

Zapobieganie zakleszczeniom

Ograniczenia sposobu zamawiania zasobów:

- **Wzajemne wykluczanie** – niepotrzebne dla podzielnych zasobów, musi być gwarantowane dla zasobów niepodzielnych.
- **Przetrzymywanie i oczekiwanie** – musi gwarantować, że proces zamawiający zasób nie posiada już jakiegoś innego zasobu.
  - Proces może zamawiać i zajmować wszystkie wymagane zasoby jedynie przed rozpoczęciem wykonania, chyba że zamawia nic nie posiadając.
  - Wady: niskie wykorzystanie zasobów, możliwość głodzenia.

Zapobieganie zakleszczeniom (c.d.)

- **Brak wywłaszczania** –
  - Jeśli proces zajmujący już jakieś zasoby chce zamówić nowe, które są aktualnie zajęte przez inny proces, to najpierw zwalnia wszystkie zajęte przez siebie zasoby.
  - Wywłaszczone zasoby są dodawane do listy zasobów na które czeka proces.
  - Proces jest wznawiane tylko wówczas, gdy może zająć jednocześnie wszystkie potrzebne zasoby.
- **Okrężne oczekiwanie** – trzeba wprowadzić uporządkowanie wszystkich typów zasobów i wymagać, by proces zajmował zasoby w tym samym porządku.

Unikanie zakleszczeń

Wymaga się, by system miał dodatkową wiedzę *a priori*.

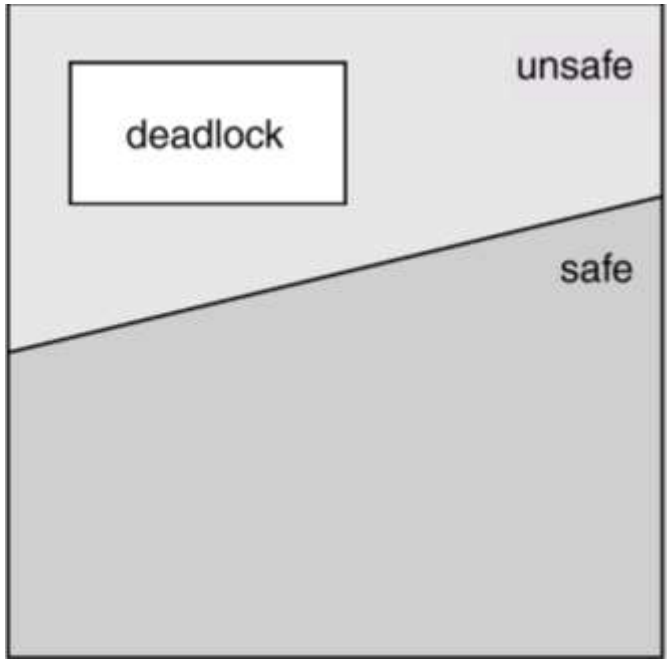
- Najprostszy i najbardziej użyteczny model zakłada, że proces musi deklarować **maksymalną liczbę** egzemplarzy zasobu każdego typu, potrzebnych w czasie wykonania.
- **Algorytm unikania zakleszczeń** dynamicznie bada stan przydziału zasobów po to, by zagwarantować, że nie dojdzie do okrężnego oczekiwania.
- **Stan przydziału zasobów** jest określony przez liczby wolnych i zajętych zasobów oraz maksymalnych zapotrzebowań na nie ze strony wykonujących się procesów.

Stan bezpieczny

- Kiedy proces żąda wolnego zasobu trzeba natychmiast ocenić czy przydział zasobu pozostawi system w stanie bezpiecznym.
- System znajduje się w **stanie bezpiecznym**, gdy istnieje **ciąg bezpieczny** dla wszystkich procesów.
- **Ciąg $<P_1, P_2, \ldots, P_n>$ procesów jest bezpieczny**, jeżeli dla każdego P_i, zamówienia zasobów, które mogą być jeszcze złożone przez P_i , mogą być zrealizowane przez obecnie dostępne zasoby oraz zasoby zajęte przez wszystkie procesy P_j, dla $j<I$.
  - Jeśli zapotrzebowanie P_i na zasoby nie może być natychmiast zrealizowane, to P_i może czekać, aż wszystkie $P_{j,,}$, $j<I$ się zakończyły.
  - Kiedy proces P_j się kończy, wówczas P_i może uzyskać potrzebne zasoby, wykonać się, zwolnić zasoby.
  - Kiedy kończy się P_i , to P_{i+1} może zająć jego zasoby itd..

Podstawowe fakty

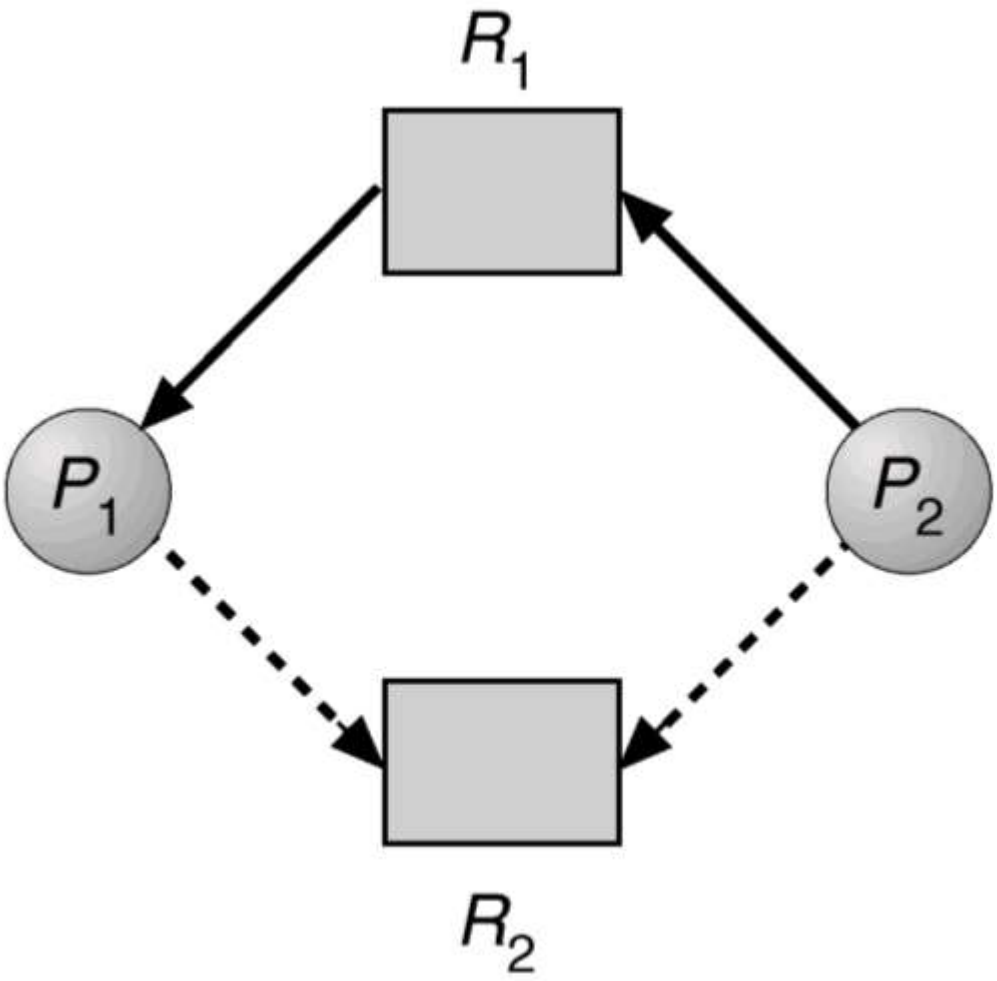
- Jeśli system jest **w stanie bezpiecznym** ⇒ **brak zakleszczeń**.
- Jeśli system **nie jest w stanie bezpiecznym** ⇒ istnieje **możliwość zakleszczenia.**
- **Unikanie zakleszczenia** ⇒ zapewnianie, że system nigdy nie wejdzie w stan niebezpieczny.



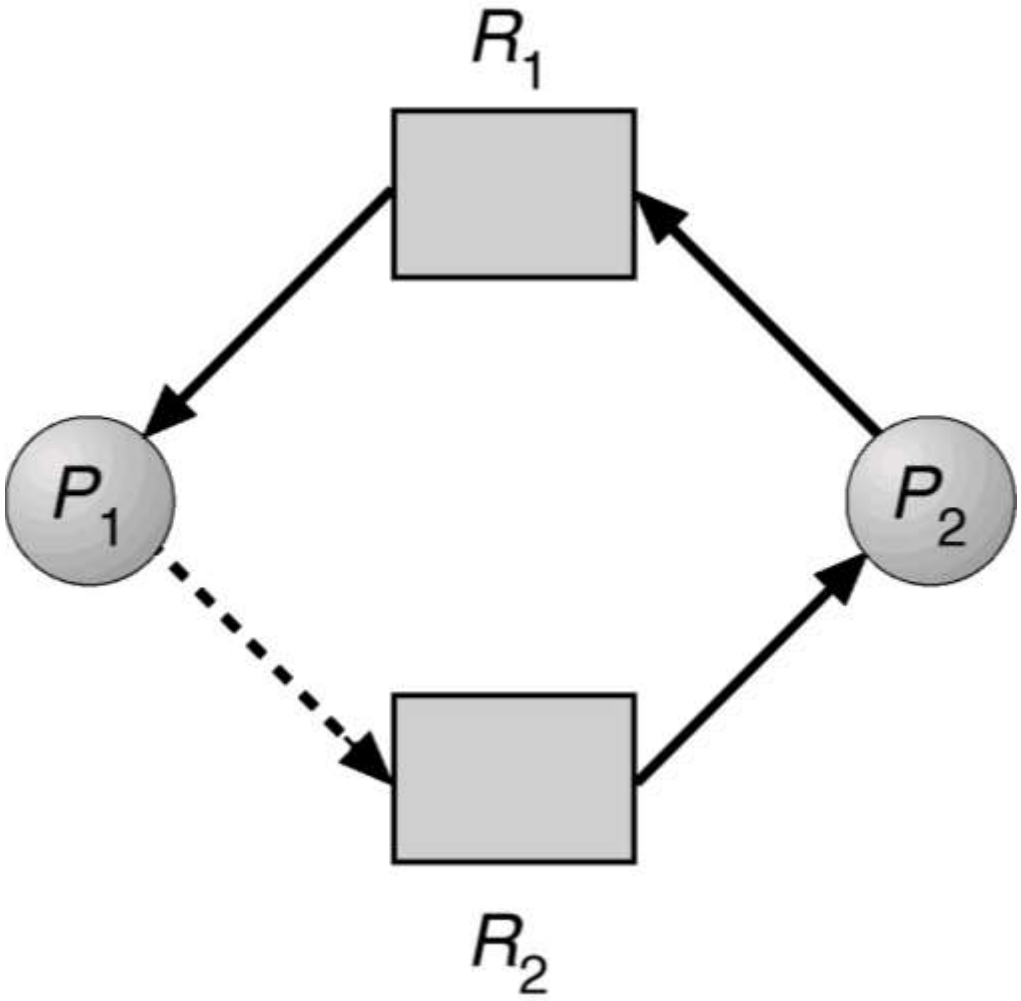
Graf alokacji (przydziału) zasobów

- **Krawędź zamówienia** $P_i \rightarrow R_j$ wskazuje, *że* proces P_j mo*że zamówić zasób* R_j; (linia przerywana).
- **Krawędź deklaracji** przechodzi w **krawędź zamówienia**, gdy proces **zamawia zasób**.
- Gdy **zasób jest zwalniany** przez proces - **krawędź zamówienia** przekształca się w **krawędź deklaracji**.
- Oświadczenia o zapotrzebowaniu na zasoby muszą być złożone przez proces **przed jego uruchomieniem**.

Graf alokacji zasobów a unikanie zakleszczeń



Stan zagrożenia w grafie alokacji zasobów



Algorytm bankiera

Założenia:

- Typy zasobów reprezentowane wielokrotnie.
- Każdy proces musi zadeklarować maksymalne zamówienia zasobów każdego typu przed rozpoczęciem wykonania.
- Proces może być zmuszony do czekania na zasób, którego zażądał.
- Kiedy proces otrzyma wszystkie potrzebne zasoby musi je zwolnić po upływie skończonego czasu.

Struktury danych algorytmu bankiera

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

- **Dostępne**: wektor o długości *m*. Jeśli *dostępne* [*j*] = *k*, to jest wolnych *k* jednostek zasobu typu R_j .
- **Maksymalne**: macierz *n x m*. Jeśli *Maksymalne* [*i,j*] = *k*, to proces P_i może zamówić najwyżej *k* jednostek zasobu typu R_j.
- **Przydzielone**: macierz *n x m*. Jeśli *Przydzielone*[*i,j*] = *k,* to proces P_i aktualnie zajął *k* jednostek zasobu typu R_j.
- **Potrzebne**: macierz *n x m*. Jeśli Potrzebne[*i,j*] = *k*, to P_i może potrzebować *k* jednostek zasobu typu R_j dla realizacji swojego zadania.

Potrzebne[i,j] = Maksymalne[i,j] – Przydzielone [i,j].

Algorytm bezpieczeństwa

1. Niech **Roboczy i Końcowy** będą wektorami o długościach *m* i *n*. Początkowo:

   Roboczy := Dostępne

   Końcowy[i] = false dla i = 1,2,3, …, n.

2. Znajdź takie *i,* że zarówno:

   (a) **Końcowy[i] = false**, jak i

   (b) **$\text{Potrzebne}_i \leq \text{Roboczy}$**

   Jeśli takie *i* nie istnieje, przejdź do punktu 4.

3. **Roboczy := Roboczy + Przydzielone_i**
   Końcowy[i] := true
   Skok do punktu 2.

4. Jeśli **Końcowy[i] = true** dla wszystkich *i*, to system **jest w stanie bezpiecznym.**

Algorytm zamawiania zasobów dla procesu P_i

Zamówienia$_i$ = wektor zamówień procesu P_i. Jeśli **Zamówienia**$_i$ **[*j*] = *k***, to proces **P**$_i$ potrzebuje **k** jednostek zasobu typu **R**$_j$.

1. Jeśli ***Zamówienia***$_i \leq$ ***Potrzebne***$_i$ przejdź do punktu 2. W przeciwnym razie zgłoś błąd, gdyż proces próbuje żądać więcej niż deklarował.
2. Jeśli ***Zamówienia***$_i \leq$ ***Dostępne***, przejdź do punktu 3. W przeciwnym razie **P**$_i$ musi czekać, gdyż nie ma wolnych zasobów.
3. Rozważ przydział żądanych zasobów procesowi ***P***$_i$ modyfikując zmienne:

   Dostępne = Dostępne - Zamówienia$_i$**;**

   Przydzielone$_i$ **= Przydzielone**$_i$ **+ Zamówienia**$_i$**;**

   Potrzebne$_i$ **= Potrzebne**$_i$ **– Zamówienia**$_{i;;}$

   - Jeśli uzyskano **stan bezpieczny** $\Rightarrow$ żądane zasoby mogą być przedzielone procesowi **P**$_i$.
   - *W przeciwnym razie* P_i *musi czekać* a poprzednie wartości zmiennych **Dostępne, Przydzielone, Zamówienia** są *odtworzone.*

Przykład działania algorytmu bankiera

- 5 procesów P_0 ,..., P_4; 3 typy zasobów: **A** (10 jednostek), **B** (5 jednostek) oraz **C** (7 jednostek).
- Migawkowy obraz systemu w chwili T_0:

	Przydzielone	Maksymalne	Dostępne
	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	

Przykład (c.d.)

- Macierz **Potrzebne** jest obliczona jako **Maksymalne – Przydzielone**.

Potrzebne

	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

- System jest w **stanie bezpiecznym**, gdyż ciąg $< P_1, P_3, P_4, P_2, P_0 >$ spełnia kryterium bezpieczeństwa.

Przykład (c.d.): P_1 zamawia (1,0,2)

- Sprawdźmy, że **Zamówione ≤ Dostępne (bo (1,0,2) ≤ (3,3,2) ⇒ true).**

	Przydzielone	Potrzebne	Dostępne
	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 1	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

- Wykonując algorytm bezpieczeństwa można pokazać, że ciąg **<P_1, P_3, P_4, P_0, P_2>** spełnia wymagania bezpieczeństwa.
- Czy zamówienie **(3,3,0)** przez **P_4** może być zrealizowane?
- Czy zamówienie **(0,2,0)** przez **P_0** może być zrealizowane?

Wykrywanie zakleszczenia

W systemie, w którym nie stosuje się algorytmu zapobiegania zakleszczeniom ani ich unikania może dojść do zakleszczenia.

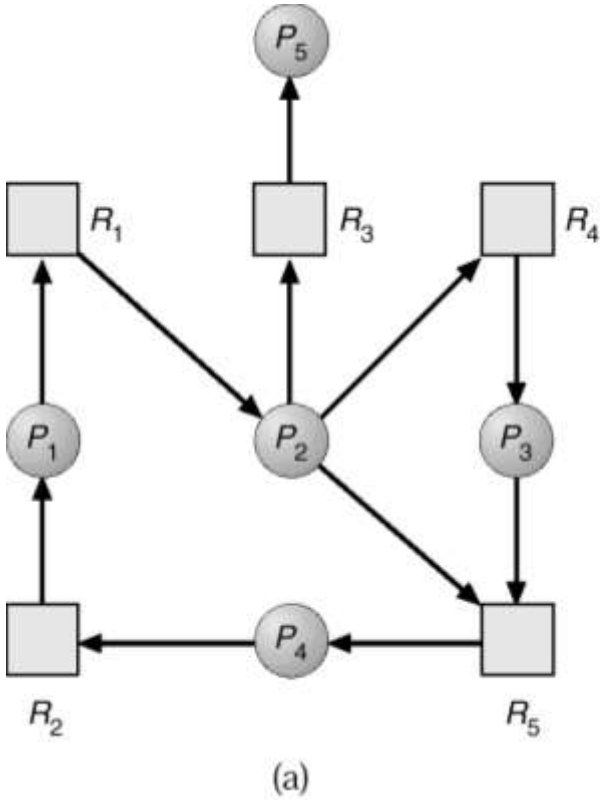
Ciąg zdarzeń:

- system wchodzi w stan zakleszczenia
- algorytm wykrywania zakleszczenia monituje o niebezpieczeństwie
- uruchamiane są procedury likwidacji zakleszczenia

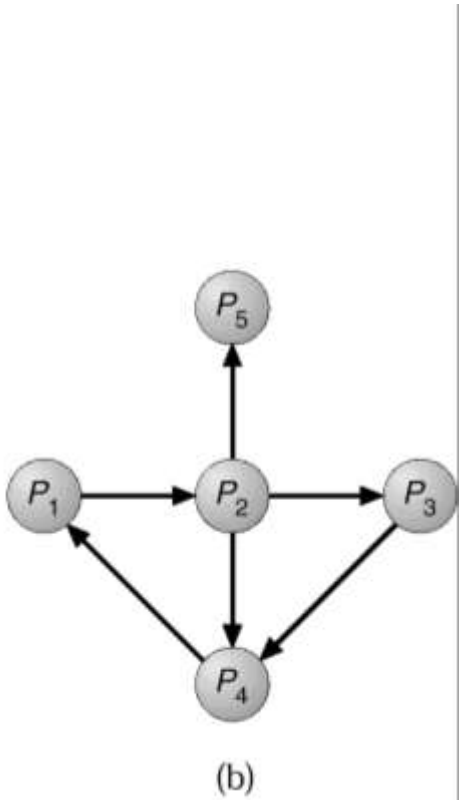
Przypadek, gdy typy zasobów są reprezentowane pojedynczo

- Można zdefiniować *graf oczekiwania* (*wait-for* graph)
  - Węzły oznaczają procesy.
  - $P_i \rightarrow P_j$ jeśli P_i oczekuje na P_j.
- Okresowo wywoływany jest algorytm sprawdzający acykliczność grafu.
- Algorytm wykrywania cyklu w grafie wymaga rzędu n^2 operacji, gdzie n jest liczbą wierzchołków grafu.

Grafy: przydziału zasobów i oczekiwania



Graf przydziału zasobów



i odpowiadający mu graf oczekiwania

Przypadek, gdy typy zasobów są reprezentowane wielokrotnie

- **Dostępne**: wektor o długości **m** zawierający liczbę wolnych jednostek zasobu każdego typu.
- **Przydzielone**: Macierz **n x m** zawierająca liczbę jednostek zasobu każdego typu przydzielona każdemu procesowi.
- **Zamówione**: Macierz **n x m** zawierająca liczbę zamówień jednostek zasobu każdego typu dla każdego procesu. Jeśli **Zamówione[ij] = k**, to proces P_i zamawia **k** dodatkowych jednostek zasobu typu R_j.

Algorytm wykrywania zakleszczeń

1. Niech ***Roboczy*** i ***Końcowy*** są wektorami o długościach ***m*** i **n**. Początkowo:
   (a) **Roboczy := Dostępne**
   (b) **For i = 1,2, …, n, if Przydzielone$_i$ ≠ 0, then Końcowy[i] := false; otherwise, Końcowy[i] := true**.
2. Znajdź indeks *i* taki, że zarówno:
   (a) **Końcowy[i] = false** jak i
   (b) **Zamówione$_i$ ≤ Roboczy**

   Jeśli nie istnieje takie *i*, to przejdź do kroku 4.

Algorytm wykrywania zakleszczeń (c.d.)

3. **Roboczy := Roboczy + Przydzielone$_i$**
   Końcowy[i] := true
   przejdź do kroku 2.
4. Jeśli ***Końcowy[i]* = false**, dla pewnych ***i*, 1 ≤ *i* ≤ *n***, to system jest w *stanie zakleszczenia*. Ponadto gdy **Końcowy[i] = false,** to **P$_i$** jest **zakleszczony**.

Algorytm wymaga rzędu *m x n*^{2} operacji dla wykrycia stanu zakleszczenia.

Przykład działania algorytmu wykrywania zakleszczeń

- Pięć procesów **P_0 ,..., P_4**; trzy typy zasobów **A** (7 jednostek), **B** (2 jednostki) oraz **C** (6 jednostek).
- Migawka stanu systemu w chwili T_0:

	Przydzielone	*Zamówione*	*Dostępne*	
		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		

- Ciąg **<P_0, P_2, P_3, P_1, P_4>** skutkuje **Końcowy[i] = true** dla wszystkich *i*.

Przykład (c.d.)

- P_2 zamawia dodatkową jednostkę zasobu typu ***C***.

Zamówione

	A	*B*	*C*
P_0	0	0	0
P_1	2	0	1
P_2	0	0	1
P_3	1	0	0
P_4	0	0	2

- Jaki jest stan systemu?
  - Nawet, jeśli odzyskać zasoby przydzielone procesowi P_0, to dostępnych zasobów nie wystarczy, aby spełnić zamówienia innych procesów.
  - Istnieje stan zakleszczenia dotyczącego procesów P_1, P_2, P_3, P_4.

Użytkowanie algorytmu wykrywania zakleszczenia

- Kiedy i jak często zależy od:
  - tego jak często zakleszczenia mają szansę powstawać,
  - ile procesów ulegnie zakleszczeniu w przypadku jego wystąpienia (koszty wycofywania)

- Jeśli algorytm detekcji będzie wykonywany w dowolnych chwilach (i rzadko), to w grafie zasobów może występować wiele cykli. Wskazanie ,,sprawcy'' zakleszczenia może być wówczas niewykonalne, a likwidowanie zakleszczenia - kosztowne.

Likwidowanie zakleszczenia: zakończenie procesu

- Zakończ wszystkie zakleszczone proces.
- Zakończ procesy zakleszczone po jednym na raz - aż do likwidacji zakleszczenia.
- W jakim porządku usuwać proces?
  - Priorytet procesu.
  - Jak długo proces się wykonywał i ile czasu potrzebuje do zakończenia.
  - Zasoby używane przez proces.
  - Zasoby potrzebne procesowi do zakończenia działania.
  - Jak wiele procesów będzie musiało być zakończonych.
  - Czy proces jest interaktywny czy wsadowy?

Likwidowanie zakleszczenia: wywłaszczenie zasobu

- **Wybór ofiary** - minimalizacja kosztu.
- **Wycofanie** (*rollback*) – powrót do pewnego stanu bezpiecznego, z którego można restartować proces.
- **Głodzenie** (*starvation*) – ciągle ten sam proces może być obierany ofiarą; należy uwzględniać koszty wycofywania.

Mieszane metody postępowania z zakleszczeniami

- Łączenie trzech podstawowych podejść:
  - **zapobieganie** (*prevention*)
  - **unikanie** (*avoidance*)
  - **wykrywanie** (*detection*)

  pozwala na znalezienie najbardziej optymalnego rozwiązania dla każdego zasobu w systemie.
- Podział zasobów na hierarchicznie uporządkowane klasy.
- Użyj najbardziej odpowiedniej techniki postępowania z zakleszczeniami dla każdej klasy.