Blokada (zakleszczenie)

Problem blokady

Zbiór zablokowanych procesów, z których każdy przetrzymuje jakieś zasoby i jednocześnie czeka na zasoby przetrzymywane przez inny proces z tego zbioru.

Przykład 1:

- System ma 2 napędy taśmy magnetycznej.
- P_1 i P_2 — każdy ma jeden z napędów i czeka na przydział drugiego.

Przykład 2:

Semafory A i B, oba zainicjowane na 1.

```
 P 0        P 1
wait(A);  wait(B)
wait(B);  wait(A)
```

Przykład 3:

Wąski, długi most na drodze.

- Ruch możliwy tylko w jednym kierunku.
- Odcinek mostu stanowi zasób.
- W przypadku blokady konieczne jest wywłaszczenie i wycofanie pojazdu lub pojazdów
- Możliwe zagłodzenie.

Model systemu

Występują zasoby różnych typów: $R_1, R_2, ...$

Zasób typu R_i może wystąpić w W_i egzemplarzach.

Każdy proces wykorzystuje zasoby w cyklu:

- żądanie,
- wykorzystanie,
- zwolnienie.

Blokada może wystąpić w przypadku zaistnienia jednocześnie następujących warunków:

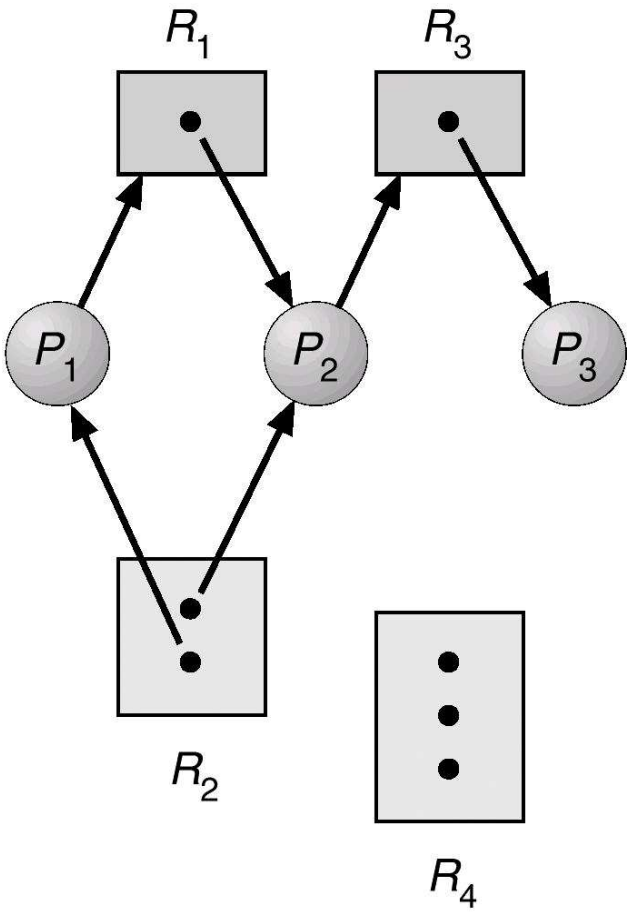
- Wzajemne wykluczanie: tylko jeden proces naraz może korzystać z zasobu.
- Przetrzymywanie i czekanie: istnieje proces przetrzymujący co najmniej jeden zasób czekający na otrzymanie zasobu przetrzymywanego przez inny proces.
- Brak wywłaszczeń: zasoby są zwracane tylko z inicjatywy procesu, który je zwalnia.
- Cykliczne oczekiwanie: istnieje zbiór procesów $P_0, P_1, ...P_n$ taki, że P_0 czeka na zasób przetrzymywany przez P_1, P_1 czeka na zasób przetrzymywany przez P_2 itd, i wreszcie P_n czeka na zasób przetrzymywany przez P_0.

Graf przydziału zasobów

Zbiór wierzchołków V i krawędzi E.

- V składa się z wierzchołków dwóch rodzajów:
  - $P = \{P_1, P_2, ..., P_n\}$ zbiór wszystkich procesów w systemie.
  - $R = \{R_1, R_2, ..., R_m\}$ zbiór wszystkich typów zasobów w systemie.

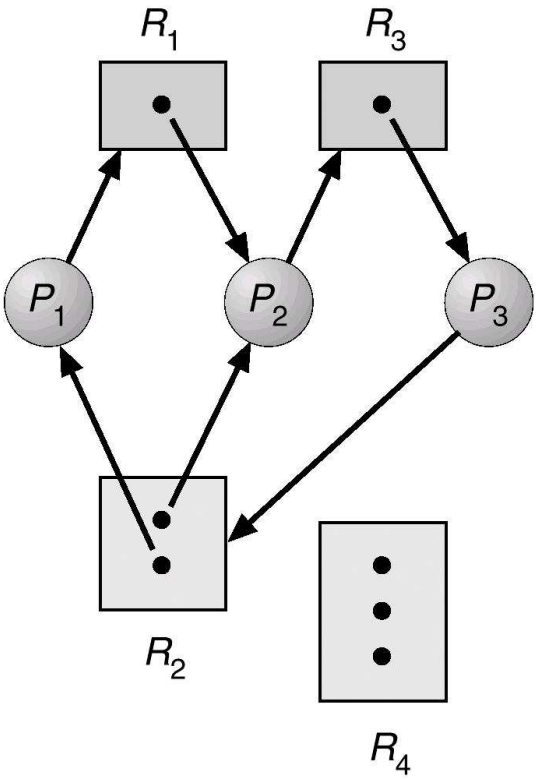
- Zamówienie — krawędź skierowana $P_i \rightarrow R_j$.
- Przydział — krawędź skierowana $R_i \rightarrow P_j$.



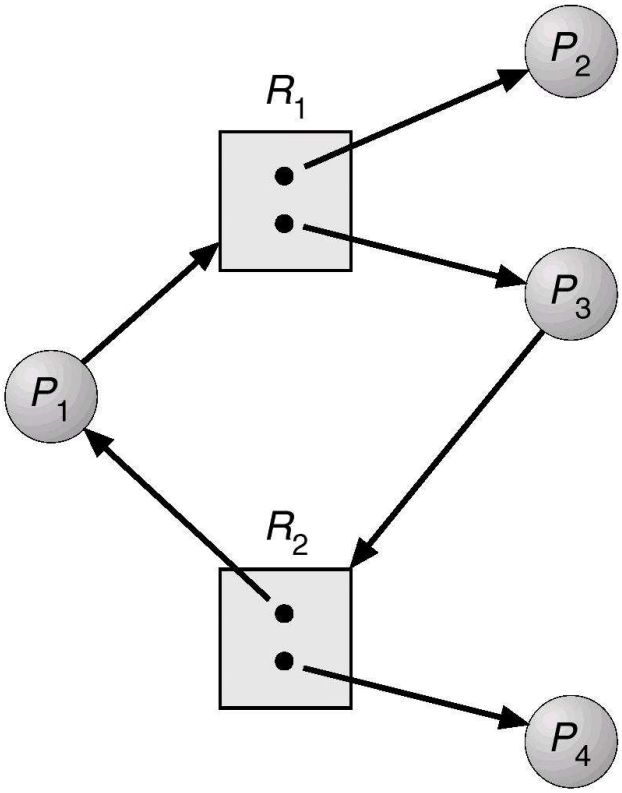
Oznaczenia:

- kółka — procesy,
- prostokąty — zasoby konkretnego typu,
- kropki w prostokatach — egzemplarze zasobu,
- krawędzie — jak wyżej.

Graf przydziału zasobów ilustrujący wystąpienie blokady.



Graf przydziału zasobów z cyklem ale bez blokady.



Podstawowe fakty:

- Jeśli w grafie nie ma cykli, nie ma blokady.
- Jeśli w grafie jest cykl, to:
  - jeśli zasoby występują w pojedynczych egzemplarzach, jest blokada;
  - jeśli zasoby występują w wielu egzemplarzach, blokada jest możliwa.

Metody postępowania z blokadami

- Postępować tak aby system nigdy nie wszedł w stan blokady — zapobieganie albo unikanie.
- Pozwalać na pojawienie się blokady, wykrywać i usuwać.
- Ignorować problem, wierząc, że blokada nie wystąpi — tak robi się w większości systemów.

Zapobieganie blokadzie

Takie postępowanie, aby co najmniej jeden z warunków koniecznych do zajścia blokady nie mógł być spełniony.

- Wzajemne wykluczanie — zależy od natury zasobu, niektóre są niepodzielne.
- Przetrzymywanie i czekanie — trzeba zagwarantować, że kiedykolwiek proces zamawia zasoby, nie przetrzymuje żadnych innych.
  - Konieczne jest zamawianie wszystkich zasobów na początku działania albo zwalnianie wszystkich przed zamówieniem następnych.

  - Słabe wykorzystanie oraz możliwość zagłodzenia.

- Brak wywłaszczania.
  - Proces, którego zamówienie nie może być natychmiast zrealizowane musi zwolnić wszystkie zasoby.
  - Zasoby zwolnione dodawane są do listy tych, na które czeka.
  - Jest wznawiany wtedy, gdy można spełnić całe jego zapotrzebowanie.
- Oczekiwanie cykliczne.

  Wprowadzamy numerację typów zasobów i żądamy aby procesy zamawiały zasoby tylko zgodnie z numeracją.

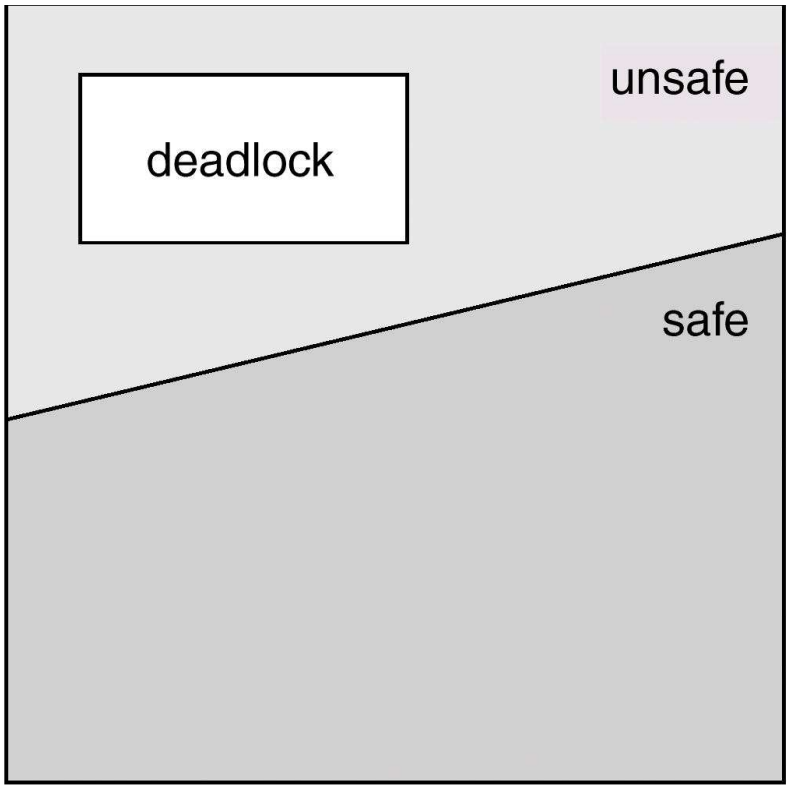
Unikanie blokad

Zakładamy, że system z góry zna sposób zamawiania zasobów przez procesy.

- Każdy proces musi zadeklarować maksymalne zapotrzebowanie na zasoby każdego typu.
- Algorytm unikania blokady dynamicznie ocenia stan przydziału zasobów i realizuje zamówienia tak, aby nie mogło dojść do cyklicznego czekania.
- Stan przydziału zasobów jest określony przez aktualne przydziały, aktualne zapasy oraz deklaracje procesów o maksymalnym zapotrzebowaniu na zasoby.

Bezpieczny stan systemu

- W momencie złożenia zamówienia system decyduje czy jego natychmiastowa realizacja pozostawi system w stanie bezpiecznym.
- System jest w stanie bezpiecznym jeśli istnieje bezpieczna sekwencja procesów.
- Sekwencja $< P_1, P_2, P_3, ..., P_n >$ jest bezpieczna jeśli dla każdego i zamówienia na zasoby, które może jeszcze złożyć proces P_i mogą zostać zrealizowane z zasobów będących aktualnie w zapasie oraz zasobów przetrzymywanych przez procesy P_j, gdzie $j < i$.
  - Jeśli zamówienie P_i nie może być natychmiast zrealizowane może on zaczeka na zakończenie wszystkich P_j.
  - Procesy P_j w momencie kończenia zwalniają wszystkie zasoby, które może otrzymać P_i.



- Jeśli taka sekwencja nie istnieje system znajduje się w stanie

zagrożenia.

Wnioski:

- Jeśli stan jest bezpieczny, nie ma blokady.
- Jeśli stan jest niebezpieczny, blokada jest możliwa.
- Utrzymując system w stanie bezpiecznym unikamy blokady.

Wykorzystanie grafu przydziału zasobów

W systemach z zasobami w pojedynczych egzemplarzach można stosować odmianę grafu przydziału zasobów.

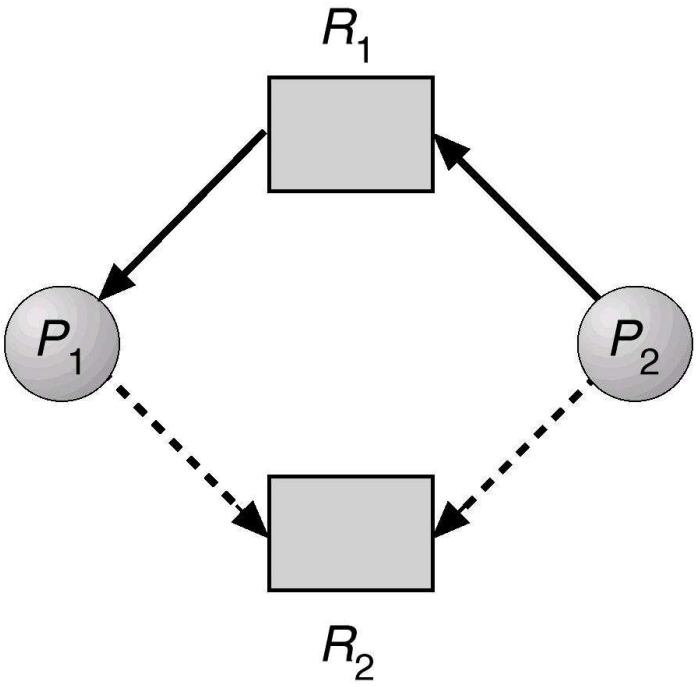
- Krawędź deklaracji $P_i \rightarrow R_j$ oznacza, że proces może zamówić zasób (linia przerywana).
- Krawędź deklaracji przechodzi w krawędź zamówienia w momencie złożenia zamówienia.
- W momencie zwolnienia zasobu krawędź przydziału przechodzi w krawędź deklaracji.

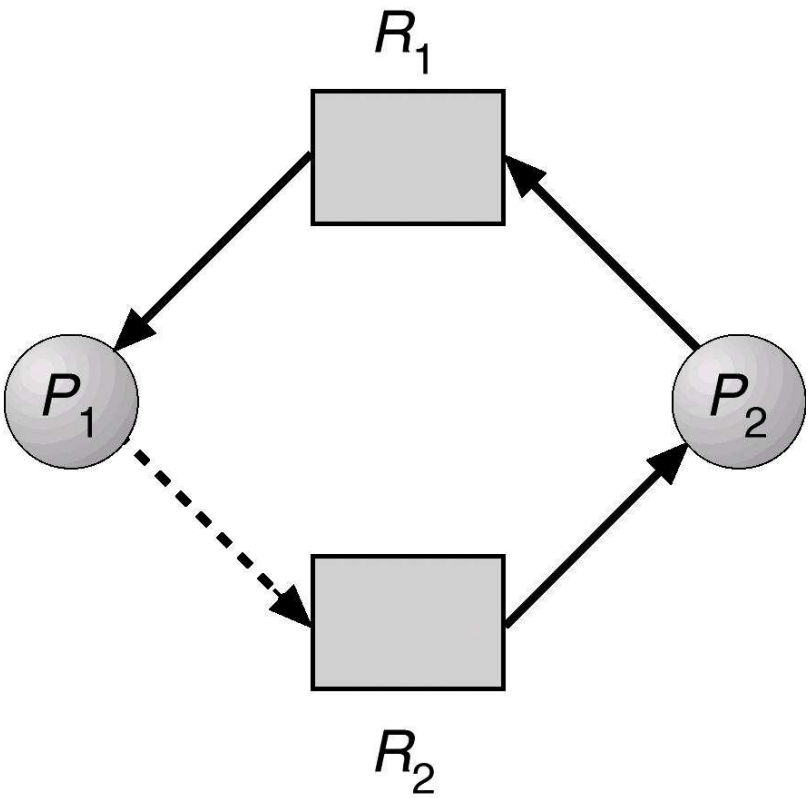
Na przykład realizacja zamówienia zasobu R_2 przez proces P_2 w następującym grafie prowadzi do stanu zagrożenia (P_1 zamawia R_2 — blokada).

Algorytm bankiera

- Zasoby w wielu egzemplarzach.
- Proces musi zadeklarować maksymalne zapotrzebowanie na zasoby.
- W razie niemożności natychmiastowej realizacji zamówienia proces musi czekać.

- Proces, który dostał wszystkie niezbędne zasoby zwróci je w skończonym czasie.





Niech n oznacza liczbę procesów w systemie a m liczbę typów zasobów.

- `Available`: Wektor długości `m`. `Available[j]=k` oznacza, że jest dostępnych `k` egzemplarzy zasobu R_j.
- `Max`: Tablica `n x m`, określająca maksymalne żądania każdego procesu. Jeśli `Max[i,j]=k`, to proces P_i może zamówić co najwyżej `k` egzemplarzy zasobu R_j.
- `Allocation`: Tablica `n x m`, określająca liczbęzasobów poszczególnych typów przydzielonych procesom. Jeśli `Allocation[i,j]=k`, wówczas proces P_i ma przydzielonych `k` egzemplarzy zasobu R_j.
- `Need`: Tablica `n x m`, określająca pozostałe do spełnienia zamówienia każdego z procesów.

  `Need[i,j] = Max[i,j] - Allocation[i,j]`.

Algorytm bezpieczeństwa

1. Niech `Work` i `Finish` będą wektorami długości `m` i `n`, odpowiednio, zainicjowanymi:

   ```
   Work = Available
   Finish[i] = false, dla i = 1,2, ..., n.
   ```

2. Szukamy takiego `i`, że zarówno

   (a) `Finish[i] = false` jak też

   (b) `Need[i,*] <= Work`.

   Jeśli takiego `i` nie ma, idziemy do punktu 4.

3. ```
   Work = Work + Allocation[i, *]
   Finish[i] = true
   ```
   ```

Idziemy do punktu 2.

4. Jeśli `Finish[i] = true` dla każdego `i`, system jest w stanie bezpiecznym.

Algorytm zamawiania zasobów

Sam algorytm bankiera jest dość prosty. W momencie złożenia zamówienia przez proces, system:

1. Sprawdza, czy zamówienie nie przekracza początkowej deklaracji procesu. Jeśli tak, zgłaszany jest błąd.
2. Sprawdza, czy zamówienie nie przekracza bieżących możliwości systemu. Jeśli tak, proces musi czekać (ponowić zamówienie później).
3. Symuluje dokonanie przydziału, odpowiednio modyfikując struktury danych i sprawdza, czy stan systemu pozostanie bezpieczny.
   - Jeśli tak, zamówienie zostaje zrealizowane.
   - Jeśli nie, proces musi ponowić zamówienie później a system przywraca stan pierwotny.

Przykład działania algorytmu Bankiera

- 5 procesów: $P_0, ..., P_4$, 3 typy zasobów: A (10 egz.), B (5egz.) C (7egz.).
- Stan systemu:

```
  Allocation  Max   Available
    A B C    A B C    A B C
P0  0 1 0    7 5 3    3 3 2
```

```
P1  2 0 0    3 2 2
P2  3 0 2    9 0 2
P3  2 1 1    2 2 2
P4  0 0 2    4 3 3
```

- Aktualna zawartość tablicy `Need`:

```
    Need
    A B C
P0  7 4 3
P1  1 2 2
P2  6 0 0
P3  0 1 1
P4  4 3 1
```

- System jest w stanie bezpiecznym. Ciąg procesów $< P_1, P_3, P_4, P_2, P_0 >$ spełnia kryteria bezpieczeństwa.
- Proces P_1 składa zamówienie `(1,0,2)`. Czy system przydzieli zasoby?
  - Dwa pierwsze warunki algorytmu przydziału są spełnione.
  - Dokonujemy przydziału próbnego:

```
    Allocation  Need   Available
       A B C    A B C   A B C
P0     0 1 0    7 4 3   2 3 0
P1     3 0 2    0 2 0
P2     3 0 1    6 0 0
P3     2 1 1    0 1 1
P4     0 0 2    4 3 1
```

  - Algorytm bezpieczeństwa pokazuje, że ciąg $< P_1, P_3, P_4, P_0, P_2 >$ spełnia kryteria.

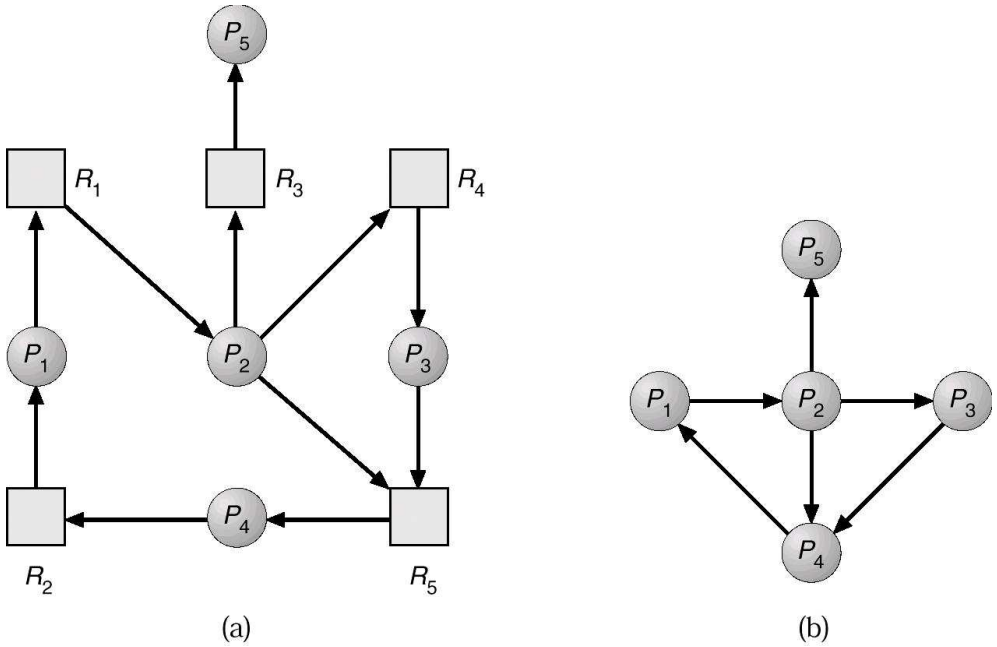
Wykrywanie blokady

Jeśli nie stosujemy algorytmów zapobiegania i unikania blokad powinniśmy mieć:

- algorytm wykrywania blokady,
- algorytm likwidowania blokady.

Typy zasobów reprezentowane pojedynczo

- Graf oczekiwania wygenerowany na podstawie grafu przydziału zasobów.
- Cykl w takim grafie oznacza blokadę (por. (b))



- Algorytm wykrywania cyklu w grafie ma złożoność n^2 (n - liczba węzłów w grafie).

Zasoby reprezentowane wielokrotnie

Do wykrywania blokady w systemie z zasobami reprezentowanymi wielokrotnie stosujemy algorytm podobny do algorytmu bezpieczeństwa (por. algorytm bankiera).

- `Available`: Wektor długości `m` określa liczbę dostępnych egzemplarzy każdego typu zasobów.
- `Request (dawniej Max)`: Tablica `n x m`, określająca bieżące zamówienia każdego procesu. Jeśli `Request[i,j]=k`, to proces P_i zamawia dodatkowo `k` egzemplarzy zasobu R_j.
- `Allocation`: Tablica `n x m`, określająca liczbęzasobów poszczególnych typów przydzielonych procesom.

Wykrywanie blokady

1. Niech `Work` i `Finish` będą wektorami długości `m` i `n`, odpowiednio, zainicjowanymi:

   ```
   Work = Available
   Dla i = 1,2, ..., n,
      jeśli Allocation[i,*] <> 0 to Finish[i] = false,
                                wpp.  Finish[i] = true,
   ```

2. Szukamy takiego `i`, że zarówno

   (a) `Finish[i] = false` jak też

   (b) `Request[i,*] <= Work`.

   Jeśli takiego `i` nie ma, idziemy do punktu 4.

3. ```
   Work = Work + Allocation[i, *]
   Finish[i] = true
   ```
   ```

Idziemy do punktu 2.

4. Jeśli istnieje `i` takie, że `Finish[i] = false`, system jest w stanie blokady i proces P_i jest w nią zaangażowany.

Złożoność algorytmu O(m x n2).

Wykorzystanie algorytmu wykrywania blokady

Decyzja kiedy i jak często wykonywać algorytm zależy od tego:

- jak często występują zakleszczenia,
- jak wiele procesów trzeba będzie ewentualnie wycofać.

Jeśli algorytm wywoływany jest w dowolnych chwilach, w grafie przydziału może być wiele cykli i ustalenie winowajcy nie zawsze jest możliwe.

Likwidowanie blokady

- Usunąć wszystkie zaangażowane procesy.
- Usunąć jeden z procesów i sprawdzać czy pomogło.
- Jak wybierać proces do usunięcia:
  - Priorytet procesu.
  - Jak długo pracuje i ile czasu potrzeba do końca.
  - Ile i jakich zasobów proces wykorzystuje.
  - Jakich zasobów brakuje procesowi.
  - Ile procesów trzeba będzie zatrzymać.
  - Czy proces jest interakcyjny czy wsadowy.

Wychodzenie przez wywłaszczenie

- Wybór ofiary. Porządek wywłaszczeń powinien minimalizować koszty.
- Wycofanie procesu do bezpiecznego punktu, zwykle do samego początku.
- Głodzenie. Jak zagwarantować, że nie będzie dochodziło do głodzenia (nie można być ofiarą zbyt wiele razy).

Metody mieszane

- Dzielimy zasoby na hierarchicznie uporządkowane klasy.
- W obrębie klasy stosujemy najodpowiedniejszą metodę.

Podsumowanie

Jaka zasada pozwoli na uniknięcie blokady w poniższej sytuacji?