Zakleszczenia

Systemy operacyjne.

Wykład 6

Wprowadzenie

- **Zakleszczenia (ang. deadlock)**
- **inna nazwa – blokady**

- w środowisku wieloprogramowym kilka procesów może rywalizować o skończoną liczbę zasobów
- proces zamawia zasoby i jeżeli nie są dostępne - wchodzi w stan oczekiwania
- oczekujący proces może nigdy nie zmienić swego stanu, gdyż ponieważ zamawiane przez niego zasoby są przetrzymywane przez inne procesy -> powstaje zakleszczenie
- uwaga: większość współczesnych systemów operacyjnych nie ma środków zapobiegania zakleszczeniom

- **Zasoby:**
  - typy zasobów, egzemplarze zasobów określonego typu

Używanie zasobu

Porządek używania zasobu przez proces:

1. **Zamówienie** -- jeżeli zamówienie nie może być spełnione (np. zasób zajęty przez inny proces), to proces zamawiający musi czekać do chwili otrzymania zasobu,
2. **Użycie --** proces może korzystać z zasobu,
3. **Zwolnienie --** proces oddaje zasób

Zasoby można zamawiać i zwalniać przez:

- wywołania funkcji systemowych
- wykonanie operacji semaforowych czekaj i sygnalizuj

Warunki konieczne zajścia zakleszczenia

Do zakleszczeń może dochodzić, gdy zachodzą **jednocześnie** następujące warunki:

1. **Wzajemne wykluczanie:** Przynajmniej jeden zasób musi być niepodzielny tzn., że zasobu tego może używać w danym czasie tylko jeden proces. Jeżeli inny proces zamawia dany zasób, to musi być opóźniony do czasu, aż zasób zostanie zwolniony.
2. **Przetrzymywanie i oczekiwanie:** Musi istnieć proces, któremu przydzielono co najmniej jeden zasób i który oczekuje na przydział dodatkowego zasobu, przetrzymywanego właśnie przez inny proces
3. **Brak wywłaszczeń:** Zasoby nie podlegają wywłaszczaniu, co oznacza, że zasób może być zwolniony tylko z inicjatywy przetrzymującego go procesu, po zakończeniu pracy tego procesu
4. **Czekanie cykliczne:** Musi istnieć zbiór {P0,P1,…, Pn} czekających procesów, takich że P0 czeka na zasób przetrzymywany przez P1, P1 czeka na zasób przetrzymywany przez proces P2,…, Pn-1 czeka na zasób przetrzymywany przez Pn, a Pn czeka na zasób przetrzymywany przez proces P0. (warunek 4 implikuje 2)

Graf przydziału zasobów

- zakleszczenie można opisać za pomocą grafu skierowanego zwanego grafem przydziału zasobów systemu (ang. system resource-allocation graph)
- graf składa się ze zbiorów wierzchołków (W) i krawędzi (K)
- zbiór wierzchołków W jest podzielony na dwa rodzaje węzłów
  - P={P1,P2,…, Pn} -- zbiór wszystkich procesów systemu
  - Z={Z1,Z2,…, Zm} -- zbiór wszystkich typów zasobów systemowych
- krawędzie
  - **krawędź zamówienia (ang. request edge)** Pi-> Zj - krawędź skierowana od procesu do typu zasobu, oznacza, że proces Pi zamówił egzemplarz zasobu typu Zj i obecnie czeka na ten zasób
  - **krawędź przydziału (ang. assignment edge)** Zj-> Pi - krawędź skierowana od typu zasobu do procesu, oznacza, że egzemplarz zasobu typu Zj został przydzielony do procesu Pi

Graf przydziału zasobów: Reprezentacja graficzna

- każdy proces Pi jest przedstawiany w postaci kółka
- każdy typ zasobu Zj jest przedstawiany w postaci prostokąta
- każdy egzemplarz zasobu jest oznaczany kropką umieszczaną w prostokącie
- krawędź zamówienia sięga do brzegu prostokąta Zj,
- krawędź przydziału musi wskazywać na jedną z kropek w prostokącie.

Graf przydziału zasobów – przykład 1

Zbiory P, Z,K

- P={P1,P2,P3}
- Z= {Z1,Z2,Z3,Z4}
- K={P1→Z1, P2→Z3, Z1→P2, Z2→P2, Z2→ P1, Z3→P3}

Egzemplarze zasobów:

- jeden egzemplarz zasobu typu Z1
- dwa egzemplarze zasobu typu Z2
- jeden egzemplarz zasobu typu Z3
- trzy egzemplarze zasobu typy Z4

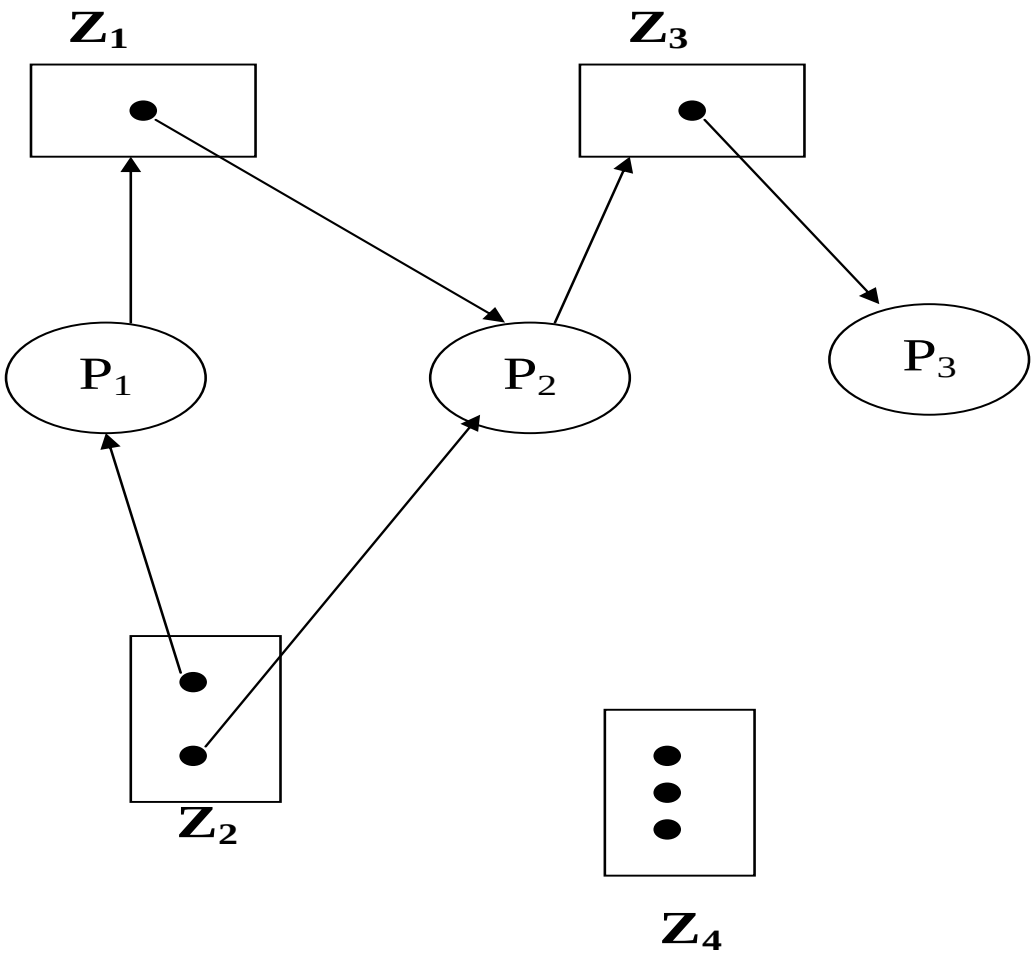
Stany procesów:

- proces P1 utrzymuje egzemplarz zasobu typu Z2 i oczekuje na egzemplarz zasobu typu Z1
- proces P2 ma po egzemplarzu Z1 i Z2 i czeka na egzemplarz zasobu typu Z3
- proces P3 czeka na egzemplarz zasobu Z3

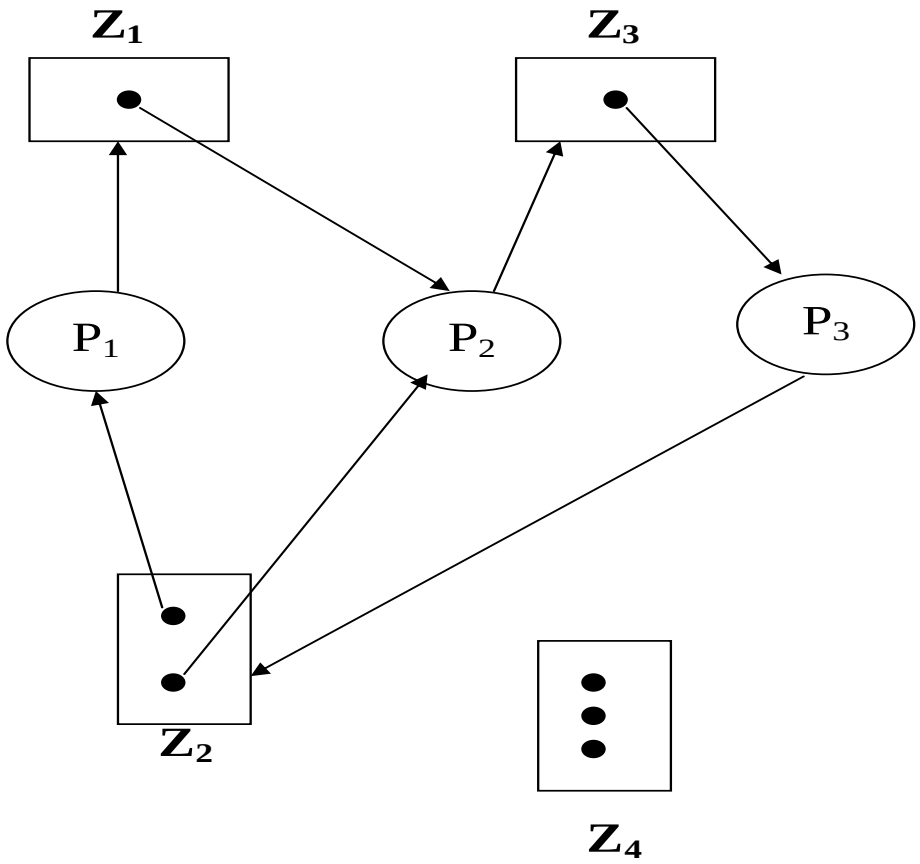
W oparciu o definicję grafu przydziału zasobów można wykazać, że:

- jeśli graf nie zawiera cykli, to w systemie nie ma zakleszczonych procesów
- jeśli graf zawiera cykl, to w systemie może dojść do zakleszczenia
- jeżeli zasób każdego typu ma tylko jeden egzemplarz, to zakleszczenie jest faktem

Graf przydziału zasobów – przykład 1



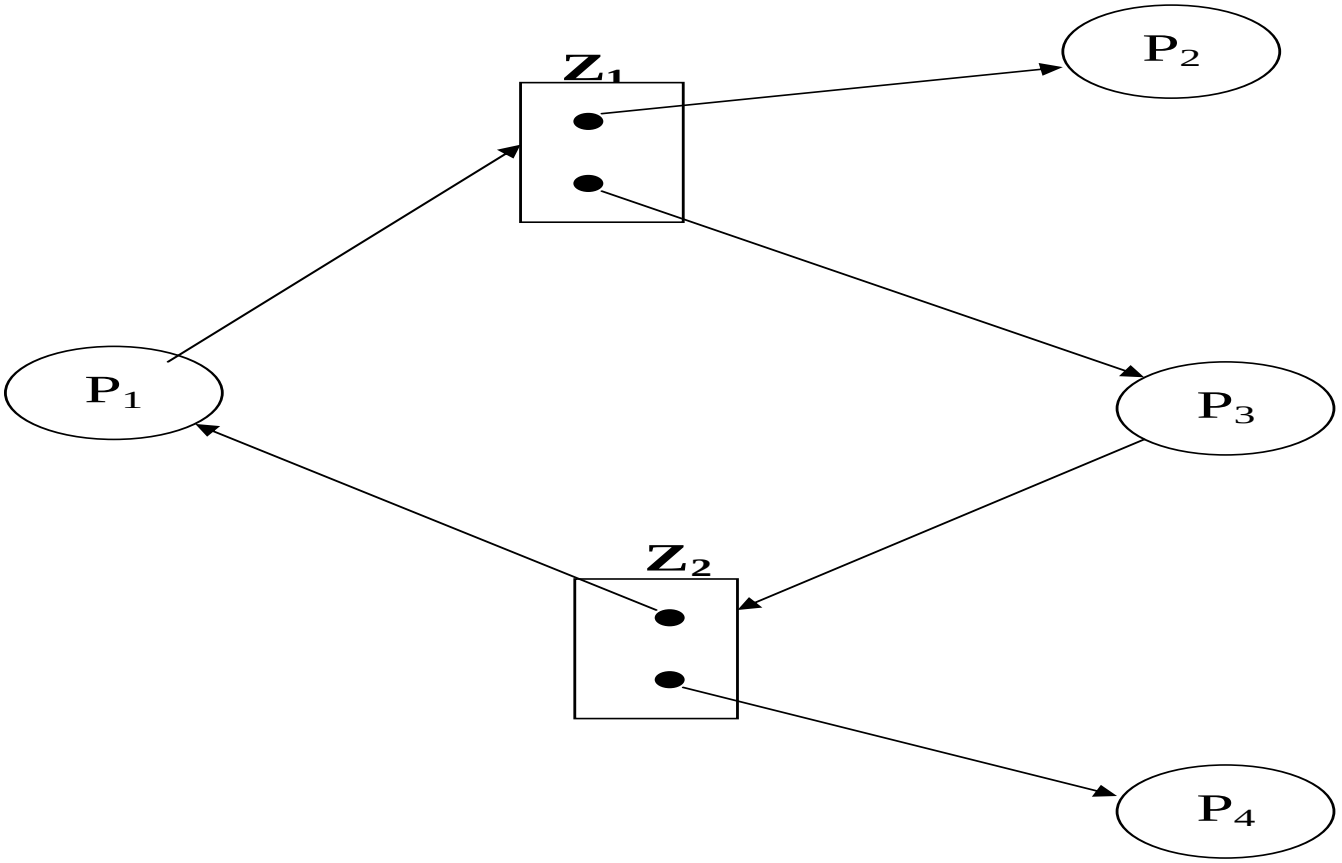
Graf przydziału zasobów – przykład 2



Istnieją cykle:

- **P1→Z1→P2→ Z3 → P3→ Z2→ P1**
- **P2→ Z3 → P3→ Z2→P2**

Graf przydziału zasobów z cyklem, ale bez zakleszczenia



Istnieje cykl: P1→Z1→P3→Z2→P1

Nie ma zakleszczenia -- proces P4 może zwolnić egzemplarz zasobu Z2, który może zostać przydzielony procesowi P3, co rozerwie cykl.

Metody postępowania z zakleszczeniami

- stosowanie protokołu, który gwarantuje, że system nigdy nie wejdzie w stan zakleszczenia
  - **zapobieganie zakleszczeniom** (ang. *deadlock prevention*) –
    - zbiór metod gwarantujących, że przynajmniej jeden warunek konieczny do wystąpienia zakleszczenia nie będzie spełniony
    - realizacja poprzez nakładanie ograniczeń na sposób zamawiania zasobów
  - **unikanie zakleszczeń** (ang. *deadlock avoidance*)
    - system operacyjny powinien dysponować dodatkowymi informacjami o zasobach, które proces będzie zamawiał i używał
    - dzięki tym informacjom można dla każdego zamówienia rozstrzygać, czy można dokonać opisanego w nim przydziału
- pozwala się systemowi na zakleszczenia, po czym podejmuje się czynności zmierzające do jego usunięcia, konieczne wtedy są:
  - wykonanie algorytmu, który sprawdza stan systemu i ocenia, że doszło do zakleszczenie
  - w razie potrzeby - wykonanie algorytmu likwidowania zakleszczenia
- można zlekceważyć ten problem, zakładając, że zakleszczenia się nie pojawią w systemie

Metody zapobiegania zakleszczeniom (1)

- gwarantuje się, że przynajmniej jeden z warunków koniecznych wystąpienia zakleszczenia nie jest spełniony
- możliwym skutkiem ubocznym jest słabe wykorzystanie urządzeń i ograniczona przepustowość systemu

Wzajemne wykluczenie

- obowiązuje tylko w przypadku niepodzielnych zasobów
- zasoby dzielone nie powodują zakleszczeń

Metody zapobiegania zakleszczeniom (2)

Przetrzymywanie i oczekiwanie:

- (1) trzeba zagwarantować, że jeżeli proces zamawia zasób, to nie powinien on mieć żadnych innych zasobów
- (2) proces zamawia i dostaje wszystkie potrzebne zasoby, zanim rozpocznie działanie
- wady powyższych 2 protokołów:
  - wykorzystanie zasobów może być niskie, gdyż wielu przydzielonych zasobów żaden proces nie będzie potrzebował przez długi czas
  - może dojść do głodzenia procesów - proces potrzebujący kilku popularnych zasobów może być odwlekany w nieskończoność, z powodu przydziału zasobów innym procesom

Metody zapobiegania zakleszczeniom (3)

Brak wywłaszczeń

należy zagwarantować wywłaszczenia -- stosuje się specjalny protokół:

- gdy proces mający jakieś zasoby zgłasza zapotrzebowanie na inny zasób, który nie może mu zostać natychmiast przydzielony, wówczas proces traci wszystkie dotychczasowe zasoby
- zwalniane zasoby są dopisywane do listy zasobów, na które proces oczekuje
- proces zostaje wznowiony, gdy można mu przywrócić wszystkie zasoby i dodać nowe, które zamawiał

Metody zapobiegania zakleszczeniom (4)

Brak wywłaszczeń (cd)

inne rozwiązanie:

- sprawdza się, czy zamawiane przez proces zasoby są dostępne
- jeżeli są - następuje przydział, jeżeli nie są - sprawdza się, czy dane zasoby są przydzielone do innego procesu, który czeka na dodatkowe zasoby - w takim wypadku odbiera mu się te zasoby i przydziela procesowi aktualnie zamawiającemu
- jeżeli zasoby nie są ani dostępne, ani przetrzymywane przez czekający proces - to proces zamawiający musi czekać
- podczas czekania proces może utracić pewne zasoby, ale tylko wtedy, jeżeli inny proces tego zażąda
- proces może być wznowiony tylko wtedy, gdy otrzyma nowe zasoby i odzyska zasoby utracone podczas czekania

Metody zapobiegania zakleszczeniom (5)

Czekanie cykliczne

- zagwarantowanie, że czekanie cykliczne nigdy nie wystąpi przez wymuszenie całkowitego uporządkowania wszystkich typów zasobów i wymaganie, by każdy proces zamawiał je we wzrastającym porządku numeracji
- Z={Z1,Z2,…, Zm} - zbiór typów zasobów
- F: Z→ N
- każdemu typowi zasobów przyporządkowuje się w sposób jednoznaczny liczbę naturalną, która umożliwia określenie, który zasób poprzedza inny w uporządkowanie

Przykład:

- F(przewijak taśmy) = 1
- F(napęd dysku) = 5
- F(drukarka) = 12
- można także wymagać, że proces zamawiający zasób typu Zj powinien mieć zawsze zwolnione zasoby Zi takie, że $F(Zi) \geq F(Zj)$

Unikanie zakleszczeń

- gromadzenie dodatkowych informacji o przebiegu zamawianiu zasobów
- opisuje się, jaka będzie kolejność zamawiania i zwalniania zasobów dla poszczególnego procesu
- system operacyjny może na podstawie tych informacji podejmować decyzje, czy po danym zamówieniu proces powinien czekać, czy nie
- system operacyjny powinien brać pod uwagę:
  - zasoby aktualnie dostępne
  - zasoby przydzielone każdemu z procesów
  - przyszłe zamawianie i zwalnianie ze strony każdego z procesów

Przykład algorytmu **unikanie zakleszczeń** (ang. *deadlock avoidance*):

- zakłada się, że każdy proces zadeklaruje maksymalną liczbę zasobów każdego typu, których mógłby potrzebować
- dysponując (z góry) dla każdego procesu informacją o wymaganej przez niego nieprzekraczalnej ilości zasobów, można zagwarantować, że system nie wejdzie w stan zakleszczenia
- algorytm sprawdza dynamicznie stan przydziału zasobów, by zagwarantować, że nie dojdzie do spełnienia warunku czekania cyklicznego
- stan przydziału zasobów jest określany przez liczbę dostępnych i przydzielonych zasobów oraz przez maksymalne zapotrzebowanie procesów

Unikanie zakleszczeń: Stan bezpieczny

- **Stan systemu jest bezpieczny (ang. safe),** jeżeli istnieje porządek, w którym system może przydzielić zasoby każdemu procesowi (nawet w stopniu maksymalnym) unikając zakleszczenia.
- System jest w stanie bezpiecznym wtedy i tylko wtedy, gdy istnieje ciąg bezpieczny.
- Ciąg procesów <P1,P2,…, Pn> jest bezpieczny w danym stanie zasobów, jeśli dla każdego procesu Pi jego potencjalne zapotrzebowanie na zasoby może być zaspokojone przez bieżąco dostępne zasoby oraz zasoby użytkowane prze wszystkie procesy Pj, gdzie j < i.
  - jeśli zasoby nie są na bieżąco dostępne, to proces Pi może poczekać, aż zakończą się wszystkie procesy Pj
  - po zakończeniu procesów Pj, proces Pi może otrzymać wszystkie potrzebne mu zasoby, wykonać swoje zadania, zwolnić zasoby i zakończyć działanie
  - po zakończeniu procesu Pi, potrzebne zasoby może uzyskać proces P_{i+1}
- Jeżeli nie istnieje ciąg procesów o podanych właściwościach, wówczas system jest w **stanie zagrożenia** (ang unsafe).
- **Stan zakleszczenia nie jest stanem zagrożenia.**
  - zakleszczenie jest stanem zagrożenia
  - nie wszystkie stany zagrożenia są zakleszczeniami, stan zagrożenia *może* prowadzić do zakleszczeń
- Zapewniając, że system będzie zawsze w stanie bezpiecznym można zagwarantować, że system nie wejdzie w stan zakleszczenia

Unikanie zakleszczeń: Algorytm grafu przydziału zasobów

- system przydziału zasobów w którym każdy typ zasobu ma tylko jeden egzemplarz
- w celu unikania zakleszczenia stosuje się odmianę przedstawionego grafu przydziału zasobów

Dodanie nowej krawędzi:

- **pozostają** krawędź zamówień i krawędź przydziałów
- wprowadza się 3 typ krawędzi - **krawędź deklaracji Pi→Zj, która oznacza, że proces Pi może zamówić zasób Zj kiedyś w przyszłości**
- krawędź deklaracji ma taki sam zwrot jak krawędź zamówienia, ale jest oznaczana linią przerywaną
- zanim proces Pi rozpocznie działanie, wszystkie jego krawędzie deklaracji muszą pojawić się na grafie przydziału zasobów (można osłabić ten warunek zezwalając, by krawędź deklaracji była dodawana do grafu wtedy, gdy wszystkie krawędzie związane z procesem Pi były krawędziami deklaracji)

- gdy proces Pi zamawia zasób Zj, wówczas krawędź deklaracji Pi→ Zj jest zamieniana na krawędź zamówienia
- gdy proces Pi zwalnia zasób Zj, wówczas krawędź przydziału Zj→ Pi jest zamieniana na krawędź deklaracji Pi→ Zj

Unikanie zakleszczeń: Algorytm grafu przydziału zasobów (2)

Rozważana sytuacja: proces Pi zawiera zasób Zj

- zamówienie może być spełnione, gdy zamiana krawędzi zamówienia Pi → Zj na krawędź przydziału Zj→ Pi nie spowoduje powstania cykli w grafie (alg. wykrywania cykli)
- jeśli powstałby cykl, to proces musi poczekać na przydział zasobu

Unikanie zakleszczeń: Algorytm bankiera

- algorytm grafu przydziału zasobów – nie nadaje się do systemu przydziału zasobów, gdzie każdy typ zasobu ma wiele egzemplarzy

- w takiej sytuacji można stosować poniższy algorytm bankiera (ang. banker's algorithm), który jest jednak mniej wydajny od schematu grafu przydziału zasobów

- pochodzenie nazwy algorytmu bankiera – mógłby służyć w systemie bankowym do zagwarantowania, że bank nie zainwestuje gotówki tak, że uniemożliwiłoby mu to zaspokojenie wymagań wszystkich jego klientów

Unikanie zakleszczeń: Algorytm bankiera

Idea algorytmu:

- gdy proces wchodzi do systemu – musi zadeklarować maksymalną liczbę egzemplarzy każdego typu zasobu, które będą mu potrzebne
- zadeklarowane liczby nie mogą przekraczać ogólnych ilości zasobów w systemie
- przy każdym zamówieniu zasobów przez proces, system określa, czy ich przydzielenie pozostawi system w stanie bezpiecznym
  - jeśli tak: zasoby zostaną przydzielone
  - jeśli nie: proces musi zaczekać, aż inne procesy zwolnią wystarczającą liczbę zasobów

Unikanie zakleszczeń. Algorytm bankiera. Struktury danych.

- **n** – liczba procesów w systemie
- **m**—liczba typów zasobów
- **Dostępne** – wektor o długości m, określa liczbę dostępnych zasobów każdego typu.
  Dostępne[j]=k oznacza, że jest dostępne k egzemplarzy zasobu Zj
- **Maksymalne** – macierz o wymiarach n x m, definiująca maksymalne żądania każdego procesu.
  Maksymalne[i,j]=k oznacza, że proces Pi może zamówić co najwyżej k egzemplarzy zasobu typu Zj
- **Przydzielone:** Macierz o wymiarach n x m definiująca liczbę zasobów poszczególnych typów, przydzielonych do każdego z procesów.
- **Przydzielone[i,j]=k** oznacza. że proces Pi ma przydzielonych k egzemplarzy zasobu typu Zj
- **Potrzebne:** Macierz o wymiarach n x m, przechowująca pozostałe do spełnienia zamówienia każdego z procesów.
- **Potrzebne[i,j]=k** oznacza, że do zakończenia swojej pracy proces Pi może jeszcze potrzebować k dodatkowych egzemplarzy zasobu Zj.
- **Potrzebne[i,j] = Maksymalne[i,j]-Przydzielone[i,j]**

Unikanie zakleszczeń. Algorytm bankiera. Założenia notacji.

- **X, Y** – wektory o długości n
- **X ≤ Y**, wtedy i tylko wtedy, gdy **X[i] ≤ Y[i]** dla każdego i=1..n
- Jeśli **Y ≤ X** i **Y≠ X**, to **Y < X**
- Wiersze w macierzach Przydzielone i Potrzebne można uważać za wektory i oznaczać:
  - Przydzielone_i – zasoby aktualnie przydzielone procesowi P_i
  - Potrzebne_i –dodatkowe zasoby, których proces P_i może jeszcze potrzebować do zakończenia zadania

Unikanie zakleszczeń. Algorytm bankiera. Algorytm bezpieczeństwa

- Algorytm rozstrzyga, czy system jest w stanie bezpiecznym, czy nie.

1. Niech Roboczy i Końcowy oznaczają wektory o długości odpowiednio m i n.
   Dokonywane są początkowe przypisania:
   Roboczy:=Dostępne;
   Końcowy[i] = false dla i=1,2... ,n

2. Znajdowane jest i takie, że:
   Końcowy[i]=false oraz $Potrzebne_i \leq$ Roboczy
   Jeśli takie i nie istnieje, to przejdź do kroku 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

5. Rząd złożoności operacji wymagany w tym algorytmie do rozstrzygnięcia o stanie bezpiecznym wynosi m x n2

Unikanie zakleszczeń. Algorytm bankiera. Algorytm zamawiania zasobów

- $Zamówienia_i$ – wektor zamówień dla procesu P_i
- $Zamówienia_i[j]$=k oznacza, że proces P_i potrzebuje k egzemplarzy zasobu typu Z_j.
- Podczas dokonywania zamówienia przez proces P_i wykonywane są następujące działania:
- 1. Jeśli $Zamówienia_i \leq Potrzebne$ – wykonaj krok 2
  W przeciwnym wypadku – sytuacja błędna, gdyż proces przekroczył deklarowane maksimum
- 2. $Zamówienia_i \leq Dostępne$, wykonaj krok 3
  W przeciwnym razie proces P_i musi czekać, ponieważ zasoby są niedostępne.
- 3. System próbuje przydzielić żądane zasoby procesowi P_i zmieniając stan w następujący sposób:
  $Dostępne := Dostępne - Zamówienia_i$;
  $Przydzielone_i := Przydzielone_i + Zamówienia_i$
  $Potrzebne_i := Potrzebne_i - Zamówienia_i$;

- Jeśli stan wynikający z przydziału zasobów jest bezpieczny, to transakcja dochodzi do skutku i proces Pi otrzymuje zamawiane zasoby.
- Jeśli nowy stan nie jest bezpieczny, to proces Pi musi czekać na realizację zamówienia Zamówienia i przywracany jest poprzedni stan przydziału zasobów

Wykrywanie zakleszczenia: Typy zasobów reprezentowane pojedynczo

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

W takiej sytuacji system powinien posiadać:

- algorytm sprawdzający stan systemu w celu wykrycia, czy wystąpiło zakleszczenie
- algorytm likwidowania zakleszczenia

- Jeśli wszystkie zasoby mają tylko po jednym egzemplarzu, wówczas można zdefiniować algorytm wykrywania zakleszczenia korzystający z odmiany grafu przydziału zasobów zwanej *grafem oczekiwania* (*ang. wait for graph*).
- Graf powstaje z grafu przydziału zasobów przez usunięcie węzłów reprezentujących typy zasobów i złączenie uwolnionych w ten sposób końców krawędzi
  - w grafie oczekiwań krawędź Pi --> Pj oznacza, że proces Pi czeka aż proces Pj zwolni potrzebne mu zasoby
  - zakleszczenie w systemie istnieje wtedy i tylko wtedy, gdy graf oczekiwania zawiera cykl

Wykrywanie zakleszczenia: Typy zasobów reprezentowane wielokrotnie

metoda grafu oczekiwań nie nadaje się do systemów, w których typy zasobów mogą mieć wiele egzemplarzy

w poniższym algorytmie korzysta się z kilku zmieniających się w czasie struktur danych analogicznych do stosowanych w algorytmie bankiera

1. Roboczy i Końcowy – wektory o długościach m i n
   Roboczy:= Dostępne
   Dla i=1...n, jeśli $Przydzielony_i \neq 0$ to Końcowy[i]= false
   w przeciwnym razie: Końcowy[i]= true

2. Odnajdywany jest indeks taki, że zachodzą dwa związki:
(a)Końcowy[i]:=false
(b) $Zamówienia_i \leq$ Roboczy
Jeśli takie i nie istnieje – idź do punktu 4.

3. Roboczy:=Roboczy+$Przydzielone_i$
   Końcowy[i]:=true
   Idź do punktu 2.

4. Jeśli dla pewnych i z przedziału $1 \leq i \leq n$ zachodzi Koncowy[i]= false, to system jest w stanie zakleszczenia
   Jeśli Koncowy[i]= false to zakleszczenie dotyczy procesu P_i

Wykrywanie zakleszczenia: Typy zasobów reprezentowane wielokrotnie (2)

Rząd operacji tego algorytmu potrzebnych do wykrycia, czy system jest w stanie zakleszczenia, wynosi m x n2

Odbierania zasobów procesowi Pi w kroku 3 dla warunku 2b:

- proces nie jest zakleszczony (2b)
- zakłada się, że nie będzie on dłużej potrzebował zasobów

Likwidowanie zakleszczeń. Zakończenie procesu

- Zakończenie procesu
- zaniechanie wszystkich zakleszczonych procesów
  - znaczny koszt (procesy mogły pracować od dawna, ich wyniki zostają zniszczone)
- usuwanie procesów pojedynczo aż do wyeliminowania cyklu zakleszczenia
  - konieczność powtarzania wykonania algorytmu wykrywania zakleszczenia po każdym usunięciu procesu
  - kryteria doboru zakańczanego procesu
    - priorytet procesu
    - jak długo działa, ile czasu jeszcze powinien się wykonywać
    - ile zasobów i jakiego typu użytkuje proces (czy można wywłaszczać te zasoby)
    - ile jeszcze zasobów proces potrzebuje do zakończenia działania
    - ile procesów trzeba będzie przerwać
    - typ procesu (interakcyjny/wsadowy)

Likwidacja zakleszczeń. Wywłaszczanie zasobów.

- wybór wywłaszczanego zasobu
  - minimalizacja ponoszonych kosztów (liczba zasobów przetrzymywanych przez proces, czas zużyty na wykonanie zakleszczonego procesu)
- wycofanie (co zrobić z procesem pozbawionym zasobu)
  - wycofanie procesu "do bezpiecznego stanu", z którego potem może być wznowiony -- konieczne byłoby przechowywanie dodatkowych informacji o dawnych stanach wszystkich procesów
  - najprostszym wyjściem -- wycofanie całkowite, wznowienie wykonania od początku
- głodzenie (czy w ten sposób nie będzie dochodziło do głodzenia procesu)
  - do roli "ofiary" może być wybierany ciągle ten sam proces (o najmniej korzystnej ocenie kosztów)

Zakleszczenia - Mieszane metody postępowania

- uznaje się, że podstawowe strategie związane z zakleszczeniami (zapobieganie, unikanie, wykrywanie) , aby zagwarantować sukces, powinny być stosowane łącznie

Opis podejścia:

- zasoby można podzielić na hierarchicznie uporządkowane klasy
- do klas stosuje się uporządkowanie typów zasobów
- dla każdej z klas dobiera się specyficzne metody postępowania z zakleszczeniami

Przykład klas zasobów (i charakterystycznych dla nich metod postępowania z zakleszczeniami):

Zasoby wewnętrzne (używane przez system, np. bloki kontrolne procesów):

- uporządkowanie zasobów

Pamięć główna

- wywłaszczenie zasobów (można przesyłać obraz zadania do pamięci pomocniczej)

Zasoby zadania (przydzielone urządzenia i pliki)

- technika unikania zakleszczeń (po uzyskaniu inf. o zapotrzebowaniu na zasoby)

Wymienny obszar pamięci (obszar w pamięci pomocniczej przeznaczony na zadania)

- wstępny przydział zasobów

Zakleszczenia – podsumowanie

Table 6.1 Summary of Deadlock Detection, Prevention, and Avoidance Approaches for Operating Systems [ISLO80]

Approach	Resource Allocation Policy	Different Schemes	Major Advantages	Major Disadvantages
Prevention	Conservative; undercommits resources	Requesting all resources at once	•Works well for processes that perform a single burst of activity •No preemption necessary	•Inefficient •Delays process initiation •Future resource requirements must be known by processes
		Preemption	•Convenient when applied to resources whose state can be saved and restored easily	•Preempts more often than necessary
		Resource ordering	•Feasible to enforce via compile-time checks •Needs no run-time computation since problem is solved in system design	•Disallows incremental resource requests
Avoidance	Midway between that of detection and prevention	Manipulate to find at least one safe path	•No preemption necessary	•Future resource requirements must be known by OS •Processes can be blocked for long periods
Detection	Very liberal; requested resources are granted where possible	Invoke periodically to test for deadlock	•Never delays process initiation •Facilitates online handling	•Inherent preemption losses