7. ZAKLESZCZENIA

Jeśli dwa pociągi zbliżają się do siebie po krzyżujących się torach, to każdy z nich powinien się zatrzymać i nie ruszać do czasu, aż drugi z nich odjedzie.

Zakleszczenie *(deadlock)*: **Oczekujące** Procesy nie zmieniają swego stanu, gdyż zamawiane przez nie Zasoby **używają** inne Procesy.

System: **skończona** liczba **Zasobów**, rozdzielanych miedzy rywalizujące ze sobą Procesy.

Każdy Proces może żądać tylu zasobów, ile potrzebuje do wykonania zadania.

Kolejność użycia zasobu przez proces:

1. **Zamówienie:** jeśli nie może być spełnione to zamawiający proces musi czekać.
2. **Użycie:** proces rozpoczyna korzystanie z zasobów.
3. **Zwolnienie:** proces oddaje zasoby.

➔ **Zasoby można Zamawiać** i **Zwalniać za pomocą funkcji systemowych.**

Przed użyciem zasobu SO sprawdza, czy Proces zamówił zasób i czy został przydzielony.

W Tablicy Systemowej zapisuje się, czy zasób jest wolny czy też przydzielony, a jeśli jest przydzielony to, do którego Procesu.

Jeśli Proces zamawia zasób, który jest **aktualnie przydzielony** innemu Procesowi, to zamawiający Proces trafia do kolejki procesów **Oczekujących** na ZASÓB.

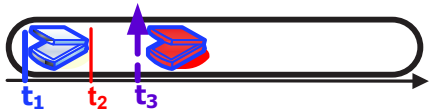
ZBIÓR procesów jest w stanie zakleszczenia, gdy każdy Proces z tego ZBIORU czeka na zdarzenie, które może spowodować tylko inny Proces z tego samego ZBIORU.

- Zakleszczenie procesów rywalizujących o zasób **tego samego typu** (**3 skanery**).

  W systemie istnieją **3 Procesy**, z których każdy przetrzymuje **jeden skaner**.

  Jeśli **każdy** Proces zamówi dodatkowy skaner, to Procesy wejdą w stan zakleszczenia.

  Każdy proces będzie **czekał** na zdarzenie **„Zwolniono skaner",** które może spowodować tylko jeden z pozostałych czekających procesów.



- Zakleszczenie przy rywalizacji o zasoby **różnych typów** (jedna **drukarka** i jeden **skaner**)

  Proces **P** ma przydzielony **skaner**, a proces **Q** przetrzymuje **drukarkę**.

  Jeśli **P** zamówi teraz **drukarkę**, a **Q** zamówi **skaner**, to wystąpi zakleszczenie.

➔ **Proces powinien natychmiast zwolnić Urządzenie po wykorzystaniu** ➔

Tylko jednoczesne zajście **4-ch warunków prowadzi** do **zakleszczenia.**

1. wzajemne wykluczanie:

przynajmniej jeden zasób ***musi być*** niepodzielny (tylko jeden proces może go używać w danym czasie).

Jeśli inny proces zamawia taki zasób, to musi Czekać do czasu zwolnienia zasobu.

2. przetrzymywanie i oczekiwanie:

istnieje proces, któremu ***przydzielono*** co najmniej jeden zasób i który oczekuje na przydział innego zasobu, przetrzymywanego przez inny proces.

3. brak wywłaszczeń:

zasoby **nie** podlegają **wywłaszczeniu** –mogą zostać zwolnione tylko przez procesy, które je przetrzymują (np. zakończenie użytkowania).

4. czekanie cykliczne:

istnieje zbiór {**P0**, **P1**, ..., **Pn**} czekających procesów

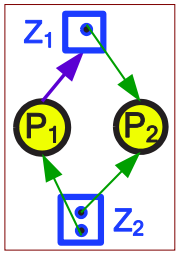
takich, że **P0** czeka na zasób przetrzymywany przez proces **P1**,

P1 ➟ P2 ➟, ..., **➟ Pn-1 ➟ Pn** oraz **Pn ➟ P0.**

➔ Warunek **czekania** cyklicznego implikuje warunek **przetrzymywania** i **oczekiwania**, więc wymienione cztery warunki nie są zupełnie niezależne.

7.1. Grafowy opis zakleszczeń

Graf przydziału zasobów systemu *(system resource-allocation graph)* jest grafem skierowanym, umożliwiającym opisywanie zakleszczenia.



W grafie występuje dwa rodzaje węzłów:

P = { P_1, P_2, ..., P_n } -zbiór wszystkich **Procesów** systemu,

Z = {Z_1, Z_2, ..., Z_n} -zbiór wszystkich **typów Zasobów** systemowych

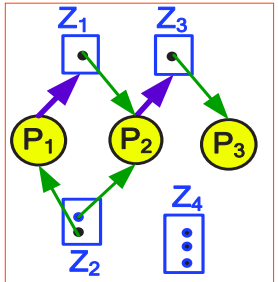
Proces **P_i** (i-ty proces) symbolizuje **kółko**.

Zasoby każdego typu **Z_k** (k-ty typ) symbolizuje **prostokąt**.

Kropki w prostokącie oznaczają liczbę ***egzemplarzy*** zasobu typu **Z_k**.

Krawędź zamówienia *(request edge)* : krawędź biegnąca od **P_i → Z_k** , dochodzi tylko do brzegu prostokąta **Z_j.**

Krawędź przydziału *(assignment edge)* : biegnie od **kropki** w prostokącie (egzemplarza zasobu) do krawędzi **koła**, krawędź **Z_k → P_i**



Rys. 7.1. Graf zasobów

$P_i \rightarrow Z_k$: proces P_i zamówił egzemplarz zasobu typu Z_k i **CZEKA** na ten zasób.

$Z_k \rightarrow P_i$: egzemplarz zasobu typu Z_k został przydzielony do procesu P_j .

Proces P_i **zamawia** zasób Z_k: w grafie przydziału zasobów umieszcza się **krawędź zamówienia**.

Jeżeli zamówienie zostaje **spełnione** to **krawędź zamówienia** zamienia się na **krawędź przydziału**.

Gdy proces zwalnia zasób to **krawędź przydziału** zostanie usunięta.

Zbiory: $\mathbf{P} = \{P_1, P_2, P_3\}$,

$\mathbf{Z} = \{ Z_1, Z_2, Z_3 \}$

$\mathbf{K} = \{ P_1 \rightarrow Z_1,\ P_2 \rightarrow Z_3,\ Z_1 \rightarrow P_2,\ Z_2 \rightarrow P_2,\ Z_2 \rightarrow P_1,\ Z_3 \rightarrow P_3 \}$

Zasoby:

1 egz. zasobu	**typu**	Z_1
2 egz. zasobu	**typu**	Z_2
1 egz. zasobu	**typu**	Z_3
3 egz. zasobu	**typu**	Z_4

Stany procesów P_1, P_2, P_3:

P_1 trzyma egzemplarz zasobu typu Z_2 i oczekuje na egzemplarz zasobu **typu** Z_1

P_2 ma egzemplarz Z_1 oraz Z_2 i czeka na egzemplarz typu Z_3

P_3 ma egzemplarz zasobu Z_3.

Można wykazać: jeśli graf nie zawiera cykli, to w systemie nie ma zakleszczonych procesów (w oparciu o definicję grafu przydziału zasobów)

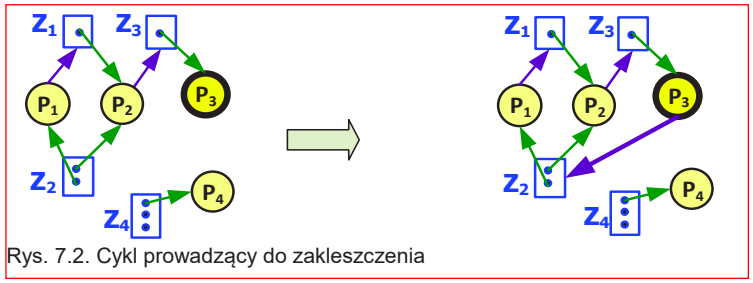
➔ **Jeśli graf zawiera cykl, to może dojść do zakleszczenia** ➔

Gdy cykl dotyczy zbioru zasobów **tylko** o jednym egzemplarzu to **jest zakleszczenie.**

Jeśli istnieje ***kilka*** egzemplarzy zasobu każdego typu, to cykl **nie** oznacza zakleszczenia.

W tym wypadku cykl w grafie jest ***warunkiem koniecznym***, lecz niewystarczającym do istnienia zakleszczenia.

Niech proces P_3 **zamawia** egzemplarz zasobu typu Z_2. (rys. 7.2)



Rys. 7.2. Cykl prowadzący do zakleszczenia

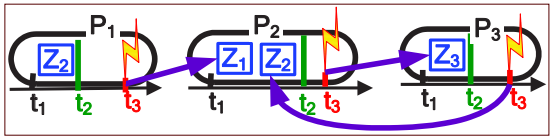
W danej chwili nie ma wolnego egzemplarza zasobu Z_2, do grafu dodaje się krawędź **zamówienia** $P_3 \rightarrow Z_2$

Od tej chwili w systemie istnieją dwa cykle:

$P_1 \rightarrow Z_1 \rightarrow P_2 \rightarrow Z_3 \rightarrow P_3 \rightarrow Z_2 \rightarrow P_1$

$P_2 \rightarrow Z_3 \rightarrow P_3 \rightarrow Z_2 \rightarrow P_2$

Procesy P_1, P_2, P_3, są **zakleszczone**.



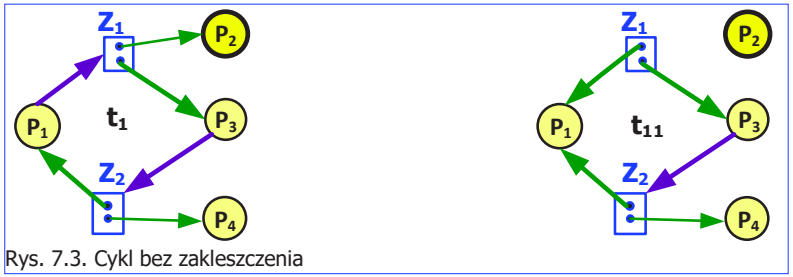
Proces P_1 czeka, aby proces P_2 zwolnił zasób Z_1.

Proces P_2 czeka, aby proces P_3 zwolnił Z_3.

Proces P_3 czeka, aby proces P_1 lub P_2 zwolnił zasób Z_2.

Procesy **nic nie robią** ⇨ zostały **usunięte** w chwili t_3 z kolejki **AKTYWNEJ**.

Na rys. 7.3. występuje cykl: $P_1 \rightarrow Z_1 \rightarrow P_3 \rightarrow Z_2 \rightarrow P_1$ **lecz niema zakleszczenia**



Rys. 7.3. Cykl bez zakleszczenia

Proces P_2 (poza cyklem) ***kiedyś*** zwolni egzemplarz zasobu typu Z_1, który przydzielony procesowi P_1, spowoduje rozerwanie cyklu.

Jeśli graf przydziału zasobów nie ma cyklu, to system nie jest w stanie zakleszczenia.

Gdy cykl istnieje system może być w stanie zakleszczenia lub nie.

7.2. Metody postępowania z zakleszczeniami

1. Zastosować protokół gwarantujący, że system **nigdy nie wejdzie** w stan zakleszczenia.

Zapobieganie zakleszczeniom:

zbiór metod zapewniających, że co **najmniej jeden** z warunków koniecznych do wystąpienia zakleszczeń nie będzie spełniony.

➔ Metody nakładają ograniczenia na sposób zamawiania zasobów.

Unikanie zakleszczeń:

SO dysponuje **dodatkowymi** informacjami o zasobach, które proces będzie zamawiał.

System - podejmując decyzję czy *aktualne* zamówienie **zrealizować** lub **odłożyć** - musi brać pod uwagę *aktualnie dostępne zasoby*.

2. **Zezwolić** na zakleszczenia, po czym usunąć je.

System dopuszczający wystąpienie zakleszczeń powinien umożliwić:

-sprawdzenie jego stanu, aby określić, czy doszło do zakleszczenia,

-zlikwidować zakleszczenie, jeśli wystąpiło.

3. Przyjąć założenie, że zakleszczenia nigdy **nie pojawią** się w systemie.

System nie testuje zakleszczeń, i będąc w stanie zakleszczenia, nie będzie o tym wiedział.

Zakleszczenie pogorszy działanie systemu, gdyż coraz więcej procesów zamawiających zasoby będzie ulegać zablokowaniu.

W wielu systemach do zakleszczeń dochodzi rzadko.

Przykładem może być proces czasu rzeczywistego wykonywany z najwyższym priorytetem (lub proces w systemie bez wywłaszczeń), który nie oddaje sterowania SO.

7.2.1. Zapobieganie zakleszczeniom

Zakleszczenie może powstać tylko po spełnieniu każdego z czterech warunków.

Gwarantując niespełnienie **przynajmniej** jednego z warunków można zapobiegać zakleszczeniom.

① Wzajemne wykluczanie

Warunek wzajemnego wykluczania musi być spełniony dla zasobów niepodzielnych.

Drukarka nie może być jednocześnie użytkowana przez kilka procesów.

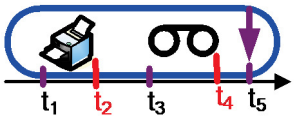
☛ Zasoby dzielone nie wymagają dostępu na zasadzie wzajemnego wykluczania, więc nie mogą powodować zakleszczeń.

Pliki udostępniane tylko do czytania są przykładem zasobu dzielonego.

Jeśli kilka procesów chce w tym samym czasie otworzyć plik do czytania, to zezwoli się im .

② Przetrzymywanie i oczekiwanie

Warunek ten nie wystąpi, jeżeli gwarantujemy, że proces zamawiający zasób, nie będzie miał żadnych innych zasobów.



Proces zamawiający zasób w chwili t_5 otrzyma go, jeżeli w chwilach t_2 i t_4 oddał wcześniej pobrane zasoby.

☛**Jeden** protokół **wymaga**, aby **każdy** proces zamawiał i dostawał **wszystkie** swoje zasoby, **zanim rozpocznie działanie**.

Wywołania funkcji systemowych realizujących **zamówienia** zasobów dla procesu muszą **poprzedzać** wywołania innych funkcji systemowych.

☛**Drugi** protokół **pozwala** procesowi na zamawianie zasobów tylko wtedy, gdy **proces nie ma żadnych zasobów.**

Zanim proces otrzyma dodatkowe zasoby, musi oddać zasoby, które ma aktualnie przydzielone.

● Proces kopiuje dane ze **streamera** do **pliku**, sortuje plik i drukuje wyniki na **drukarce**.

W pierwszej metodzie proces ***na wstępie*** zamawia streamer, plik dyskowy i **drukarkę**.

Będzie zajmował **drukarkę** przez czas swojego działania, mimo że potrzebuje ją na końcu.

W drugiej metodzie proces zamówi na początku ***tylko*** streamer i plik dyskowy.

Proces przekopiuje dane z taśmy na dysk, po czym ***zwolni*** streamer i plik dyskowy.

✝W następnej kolejności proces musi ***ponownie zamówić*** plik dyskowy oraz drukarkę.

Po wydrukowaniu pliku z dysku na drukarce proces zwalnia oba zasoby.

Wady obu protokołów:

1. prawdopodobieństwo **małego wykorzystania zasobów**, ponieważ z wielu zasobów **przydzielonych jednemu procesow**i nie będą mogły korzystać inne procesy przez pewien okres czasu.
2. może dojść do **głodzenia**; proces potrzebujący **kilku zasobów** może być odwlekany z powodu ciągłego przydzielania jednego z nich innym procesom.

③ Brak wywłaszczeń

● Proces **ma zasoby** i **zgłasza zapotrzebowanie** na **nowy zasób**, którego **nie można** natychmiast przydzielić (proces musiałby czekać).

Wówczas proces **traci wszystkie** dotychczasowe zasoby.

Zasoby **zwalniane** są niejawnie i **dopisywane** do listy zasobów, których proces oczekuje.

Proces zostanie wznowiony, gdy będzie można przywrócić mu jego **dawne** zasoby oraz dodać **nowe**, które zamawiał.

● Proces **zamawia zasoby**: **najpierw sprawdza się,** czy są one **dostępne**.

Jeśli tak, to zostają mu przydzielone.

Jeśli zasoby **nie** są dostępne, to sprawdza się, czy dane zasoby przydzielone są do ***innego*** procesu (aktualnie przez niego przetrzymywane), ***który czeka*** na **dodatkowe** zasoby.

Jeśli tak, to ***innemu*** *(oczekującemu)* **odbiera** się zasoby i przydziela procesowi aktualnie zamawiającemu.

Jeżeli zasoby nie są przetrzymywane przez ***inny*** czekający proces to proces **zamawiający** musi czekać.

Czekając proces **może utracić** pewne zasoby, gdy inny proces ich zażąda.

Proces zostanie wznowiony tylko wtedy, gdy otrzyma nowe zasoby i **odzyska** zasoby utracone podczas oczekiwania.

④ Czekanie cykliczne

Czekanie cykliczne nie wystąpi, po uporządkowaniu wszystkich typów zasobów i zamawianiu ich przez procesy we wzrastającym porządku ich numeracji.

Niech **Z** = { Z_1, Z_2,..., Z_n} będzie zbiorem **typów** zasobów.

☛ Każdemu **typowi zasobu** przyporządkowuje się, w sposób przemyślany, liczbę całkowitą.

Definiuje się funkcję **F: Z ← N**, gdzie N jest zbiorem liczb naturalnych.

Jeżeli zbiór **Z** = { streamer, napęd dysków, drukarka } to funkcję **F** można zdefiniować:

F(streamer) = 1

F(napęd dysku) = 5

F(drukarka) = 12

☛ Funkcja **F** determinuje naturalny porządek używania zasobów w systemie.

Każdy proces może zamawiać zasoby tylko we wzrastającym porządku ich numeracji.

Proces może zamówić początkowo dowolną liczbę egzemplarzy zasobu typu Z_i.

Potem proces może zamówić egzemplarze zasobu typu Z_k wyłącznie, gdy $F(Z_k) > F(Z_i)$.

☛ Kilka egzemplarzy zasobu ***tego samego typu*** zamawia się jednym zamówieniem.

> Proces, który chce używać **jednocześnie streamer** i **drukarkę**, musi **najpierw zamówić streamer**, a **potem drukarkę**.

Można dowieść, że stosując powyższy protokół, warunek czekania cyklicznego nie wystąpi.

7.2.2. Unikanie zakleszczeń

Ubocznym skutkiem zapobiegania zakleszczeniom może być słabe wykorzystanie urządzeń i ograniczona przepustowość systemu.

☛Metoda unikania zakleszczeń wymaga **dodatkowych** informacji ***o sposobie*** zamawianie zasobów.

- Dany jest system z jednym **streamerem** i jedną **drukarką**.

  Proces **P** chce najpierw zamówić **streamer**, a następnie **drukarkę**, zanim zwolni oba zasoby.

  Proces **Q** potrzebuje najpierw **drukarki**, a potem **streamera.**

Informacje o kolejności zamówień i zwolnień dla każdego procesu, pozwalają SO decydować przy każdym zamówieniu, czy proces powinien czekać, czy nie.

System musi wziąć pod uwagę: -zasoby dostępne bieżąco,
-zasoby już przydzielone każdemu procesowi;
-przyszłe zamówienia i zwolnienia ze strony każdego procesu.

Algorytm unikania zakleszczenia *(dead lock avoidance)*
sprawdza **dynamicznie** stan przydziału zasobów, aby zagwarantować, że nigdy **nie dojdzie** do spełnienia warunku czekania cyklicznego.

Każdy proces deklaruje **maksymalną** liczbę potrzebnych zasobów **każdego typu**.

Stan przydziału zasobów określa: -liczba dostępnych i przydzielonych zasobów,
-maksymalne zapotrzebowania procesów.

❒ **Stan bezpieczny**

Stan systemu jest bezpieczny jeśli istnieje porządek, w którym system może przydzielać zasoby każdemu Procesowi stale unikając zakleszczenia.

Ciąg < $P_1, P_2,...,P_i,...,P_k...,P_n$ > jest bezpieczny, jeśli dla każdego procesu P_k jego zapotrzebowanie może być zaspokojone przez zasoby:
-bieżąco dostępne,
-użytkowane przez procesy P_i, dla **i < k** (wcześniejsze).

Jeśli ciąg taki nie istnieje, to system jest w stanie zagrożenia.

Jeśli zasoby dla procesu P_k nie są natychmiast dostępne, to może on poczekać, aż zakończą się wszystkie procesy P_i.

Gdy procesy P_i zakończą się to proces P_k może otrzymać potrzebne zasoby.

Kiedy proces P_k będzie zakończony, wtedy niezbędne zasoby może otrzymać proces P_{k+1} itd.

† Zakleszczenie jest stanem zagrożenia †

Nie wszystkie stany zagrożenia są zakleszczeniami.

☛ **Stan zagrożenia może prowadzić do zakleszczenia.**

> W stanie **zagrożenia** SO nie może zapobiec zamówieniom procesów, prowadzących do zakleszczenia.
>
> ➔ Zachowanie procesów steruje stanami zagrożenia.

Przykład: system składa się z **12**-stu skanerów i **3** procesów: **P0, P1, P2** .

Zapotrzebowanie maksymalne: **P0 → 10**; **P1 → 4**; **P2 → 9**.

W chwili t_0: **P0** ma **5** skanerów
P1 ma **2**
P2 ma **2**
∑9 czyli 3 wolne

W chwili t_0 system z ciągiem procesów **<P1, P0, P2>** jest w stanie bezpiecznym.
Proces **P1** otrzyma natychmiast **4** skanery, a kiedy je zwróci (będzie **5** wolnych),
wtedy **P0** otrzyma komplet skanerów i kiedy je zwolni (będzie **10** wolnych),
wtedy **P2** otrzyma wszystkie potrzebne skanery.

➔ **W** chwili t_1 proces **P2** otrzymał dodatkowy skaner;
system nie jest już w stanie bezpiecznym, gdyż tylko proces **P1** otrzyma wszystkie potrzebne skanery.

<P1, P0, P2> W chwili t_1: **P0** ma **5** // potrzebuje 4
P1 ma **2** // potrzebuje 4
P2 ma **3** // potrzebuje 9
∑10 czyli 2 wolne

Proces **P1** otrzyma natychmiast **4** skanery, a kiedy je zwróci (będzie **4** wolnych),
wtedy **P0** (ma **5**) po zamówieniu dalszych **5** będzie musi czekać (nie są dostępne),
następnie **P2** (ma **2**) po zamówieniu **6** będzie musiał czekać, i mamy **zakleszczenie**.

Popełniony błąd: zgoda na przydzielenie procesowi **P2** w chwili t_1 jeszcze jednego skanera.

> Gdyby proces **P2** **czekał** dopóki nie zakończy się któryś z pozostałych procesów i nie odda swoich zasobów, wówczas można byłoby **uniknąć** **zakleszczenia**.

Algorytmy unikania zakleszczeń opierają się na idei zapewnienia, że system będzie stale pozostawał w stanie bezpiecznym.

Stan początkowy systemu jest bezpieczny.

Proces zamawia zasób, **który jest na bieżąco dostępny**, system decyduje, czy zasób wolno przydzielić **natychmiast**, czy proces powinien **poczekać**.

Zgoda jest udzielona tylko wtedy, kiedy przydział pozostawia system w stanie bezpiecznym.

☛ **Proces może oczekiwać na zasób nawet, gdy jest aktualnie dostępny.**

Może to zmniejszyć wykorzystanie zasobów w stosunku do rezygnacji z algorytmu unikania zakleszczenia.

❒ **Zmodyfikowany Graf przydziału zasobów**

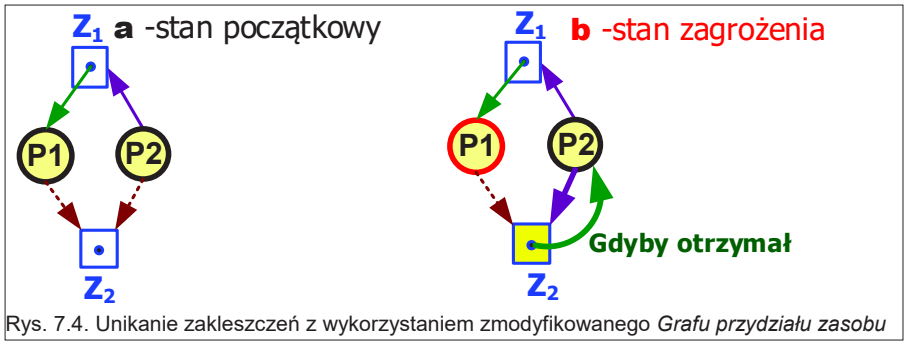
Zmodyfikowany *Graf przydziału zasobów* pozwala unikać zakleszczenia, gdy w systemie każdy typ zasobu ma **tylko jeden** egzemplarz.

Krawędź deklaracji *(claim edge)* -linia przerywana, zwrot jak krawędź zamówienia.

Krawędź deklaracji P_i -----> Z_k wskazuje, że
proces P_j **może** zamówić zasób Z_k w **przyszłości**.

Gdy proces P_i zamawia zasób Z_k to krawędź **deklaracji** zamienia się na krawędź **zamówienia**.

Gdy proces P_i zwalnia zasób Z_k to krawędź **przydziału** Z_k → P_i jest zamieniana z powrotem na krawędź **deklaracji** P_i -----> Z_k



Rys. 7.4. Unikanie zakleszczeń z wykorzystaniem zmodyfikowanego *Grafu przydziału zasobu*

Proces P_i zamawia zasób Z_k.

Zamówienie może być spełnione tylko wtedy, gdy zamiana krawędzi zamówienia P_i → Z_k na krawędź **przydziału** Z_k → P_i nie utworzy **cyklu** w grafie przydziału zasobów.

☛ Sprawdzenie bezpieczeństwa: wykonanie algorytmu wykrywania cyklu w grafie przydziału.
Jeśli nie ma cyklu, to przydział zasobu pozostawi system w stanie bezpiecznym.

> Gdyby **znaleziono cykl**, wykonanie przydziału wprowadziłoby system w stan **zagrożenia**.
> Dlatego proces P_i, musi czekać na spełnienie swoich zamówień.

Proces P_2 zamawia zasób Z_2 (rys. 7.4).

Zasób Z_2 jest wolny, ale **nie można** przydzielić go procesowi P_2, gdyż **spowodowałoby** to powstanie ***cyklu w grafie***.

Cykl wskazywałby, że system jest w stanie **zagrożenia**.

Jeżeli proces P_1 zgłosiłby zamówienie na zasób Z_2, to nastąpiłoby **zakleszczenie.**

☛ **Deklaracje o zapotrzebowaniu na zasoby muszą być złożone w systemie wcześniej.**

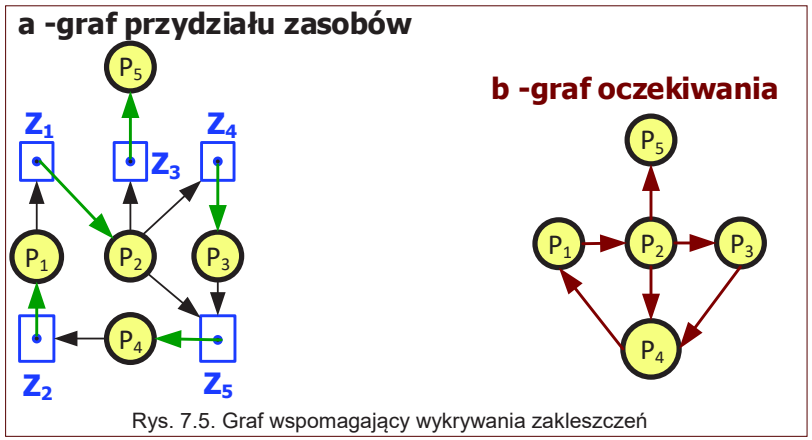
Zanim proces P_i rozpocznie działanie, wszystkie jego krawędzie **deklaracji** muszą pojawić się w grafie przydziału zasobów.

> Można zezwolić, aby krawędź **deklaracji** P_i---> Z_k była dodawana do grafu tylko wtedy, gdy ***wszystkie*** krawędzie związane z procesem P_i są *krawędziami* ***deklaracji***.

7.2.3. Wykrywanie zakleszczenia

Graf oczekiwania *(wait_for graph)* jest algorytmem wykrywania zakleszczeń, gdy wszystkie zasoby mają tylko po **jednym egzemplarzu**.

Wykorzystuje zmodyfikowany graf przydziału zasobów, powstały przez **usunięcie** węzłów reprezentujących **typy** zasobów i **złączenie** uwolnionych końców krawędzi.



Rys. 7.5. Graf wspomagający wykrywania zakleszczeń

W **grafie oczekiwania** krawędź od P_i → P_k implikuje, że proces P_i czeka aby proces P_k zwolnił potrzebne mu **zasoby**.

Krawędź P_i → P_k istnieje w grafie **oczekiwania** wtedy i tylko wtedy, gdy odpowiadający mu graf przydziału zasobów zawiera dwie krawędzie:

$P_i \rightarrow Z_q$ i $Z_q \rightarrow P_k$ dla pewnego zasobu Z_q.

Zakleszczenie w systemie wystąpi wtedy i tylko wtedy, gdy graf oczekiwania zawiera cykl.

Wykrywanie zakleszczeń wymaga aby system utrzymywał **graf oczekiwania** i okresowo wykonywał algorytm wykrywania cykli w grafie.

Rząd operacji algorytmu wykrywania cykli w grafie wynosi n^2, **n** -liczba wierzchołków grafu.

Przedstawiony Graf oczekiwania nie nadaje się, gdy istnieje wiele egzemplarzy danego typu zasobu.

Algorytm wykrywania zakleszczenia, w tym przypadku, wykorzystuje zmieniające się w czasie struktury danych, aby badać wszystkie możliwe ciągi przydziałów dla procesów.

Dostepne[1..m] -liczba dostępnych **zasobów** każdego typu.

Przydzielone[1..n,1..m] -liczba zasobów każdego typów, przydzielonych do każdego procesu.

Zamowienia[1..n,1..m] -określa bieżące zamówienie każdego procesu.

Rząd operacji potrzebnych do wykrycia, czy system jest w stanie zakleszczenia wynosi **m** x n^2.

Kiedy należy wywoływać algorytm wykrywania zakleszczenia?

1. Zależnie od **częstości** wystąpienia zakleszczeń.
2. Zależnie od **liczby procesów** ulegających zakleszczeniu, gdy zakleszczenie wystąpi.

Jeśli zakleszczenia występują **często**, to algorytm wykrywania należy wywoływać **często**.

Zasoby przydzielone zakleszczonym procesom pozostają bezużyteczne.

Liczba zakleszczonych procesów może rosnąć.

- Groźba zakleszczeń występuje tylko wtedy, gdy proces zgłasza zamówienie, które nie może być **natychmiast** zrealizowane.
  Takie zamówienie może być finalną potrzebą, której zaspokojenie kończy łańcucha czekających procesów.
- Algorytm wykrywania zakleszczenia może być wywoływany każdorazowo, gdy zamówienie na przydział nie może być spełnione **natychmiast**.
  Można wtedy zidentyfikować zarówno zbiór zakleszczonych procesów jak też proces, który do tego doprowadził.

Wykrywanie zakleszczeń przy każdym zamówieniu może prowadzić do wydłużenia czasu obliczeń.

Mniej kosztowna metoda polega na rzadszym wykonywaniu algorytmu.

Na przykład każdorazowo wówczas, gdy wykorzystanie CPU spadnie poniżej 40%.

Zakleszczenie, prędzej czy później powoduje spadek wykorzystania CPU.

- Jeśli algorytm wykrywania wykonywany jest w dowolnych chwilach, to w grafie zasobów można często odnotować wiele cykli.
  Wskazanie pośród wielu zakleszczonych procesów „sprawcy" zakleszczenia może być niewykonalne.

7.3. Likwidowanie zakleszczenia

1. Usuwanie procesów w celu przerwania czekania cyklicznego.

- **Zaniechanie wszystkich zakleszczonych procesów.**
  ⬅ Metoda rozrywa cykl zakleszczenia ➡
  Koszt metody może być duży, gdyż likwidowane procesy mogły działać od dawna, a ich wyniki częściowe zostaną zniszczone.
  Problem: Jeśli proces uaktualniał dane w pliku, to nagłe zakończenie go pozostawiłoby plik w nieokreślonym stanie.
- **Usuwanie procesów pojedynczo,** aż do **wyeliminowania** cyklu **zakleszczenia.**
  Konieczność powtarzania algorytmu ***wykrywania zakleszczenia*** po każdym usunięciu procesu w celu sprawdzenia, czy pozostałe procesy nadal są zakleszczone.
  Trzeba rozstrzygać, który proces należy zakończyć. Jest to decyzja polityczna.
  Należy zaniechać te procesy, których przedwczesne zakończenie ***zminimalizuje koszty***.
  Jaki jest priorytet procesu ?
  Ile zasobów i jakiego typu użytkuje proces ?
  Ile procesów trzeba będzie przerwać ?
  Czy proces jest interakcyjny czy wsadowy ?
  Ile czasu potrzebna procesowi do zakończenia obliczeń ?
  Ile zasobów proces potrzebuje do zakończenia działania ?

2. Odebranie pewnych zasobów procesom, których dotyczy zakleszczenie.

Stopniowo odbiera się zasoby jednym procesom i przydziela innym, aż do przerwania cyklu zakleszczenia.

Stosując wywłaszczenie należy uwzględnić **trzy** kwestie:

Wybór ofiary: Które zasoby i procesy mają ulec wywłaszczeniu?
Porządek wywłaszczeń powinien minimalizować koszty.
Koszty to: **liczba zasobów**, które przetrzymuje zakleszczony proces,
czas, który zakleszczony proces **dotychczas** zużytkował.

Wycofanie: Co zrobić z procesem, który zostanie wywłaszczony z zasobu?
Trzeba wycofać proces do bezpiecznego stanu, z którego można go będzie wznowić.
Trudno określić, czym byłby stan bezpieczny.
☛ Najprościej jest zaniechać proces i później wykonać go od **początku**.
Korzystniej wycofać proces **do moment** niezbędnego dla zlikwidowania zakleszczenia.

Głodzenie: Jak zagwarantować, że nie będzie dochodzić do głodzenia procesu?
tzn. że wywłaszczanie nie będzie dotyczyć stale tego samego procesu?

☛ **Jeżeli wybór ofiary opiera się na ocenie kosztów**, to może być wybierany wciąż ten sam proces; proces nigdy nie zakończy swojej pracy - zjawisko ***głodzenia***.
Proces powinien być wydelegowany do roli ofiary skończoną liczbę razy.

Stosowane indywidualnie podstawowe strategie postępowania z zakleszczeniami (zapobieganie, unikanie i wykrywanie), **nie rozwiązują wszystkich problemów przydzielania zasobów.**

❒ **Wszystkie zasoby można podzielić na hierarchicznie uporządkowane klasy.**

Do każdej klasy stosuje się **indywidualną**, lecz typową technikę porządkowania zasobów.

➔ **W obrębie klas można dobrać najlepsze metody postępowania z zakleszczeniami.**

System, w którym zastosuje się tę strategię, nie będzie narażony na zakleszczenia.
Dzięki uporządkowaniu zasobów zakleszczenia mogą ograniczyć się do jednej klasy.
Wewnątrz danej klasy można zastosować jedną z podstawowych metod.

Przykład czterech klas zasobów z **indywidualnymi** technikami zapobiegania zakleszczeniom:

1. **Zasoby wewnętrzne** -używane przez system, takie jak bloki kontrolne procesów.
   Uporządkowanie zasobów, gdyż nie trzeba dokonywać wyborów między realizowanymi zamówieniami.
2. **Pamięć główna** -używana przez zadanie użytkownika.
   Wywłaszczanie, ponieważ zawsze można przesłać obraz zadania do pamięci dyskowej, co pozwala na wywłaszczanie procesów z pamięci głównej.
3. **Zasoby zadania** -przydzielane urządzenia w rodzaju ***drukarka*** czy ***pliki***.
   Unikanie zakleszczeń, ponieważ niezbędne informacje o zapotrzebowaniu na zasoby mogą być uzyskane z kart sterujących zadania.
4. **Wymienny obszar pamięci** -obszar w pamięci dyskowej, dla zadań użytkowników.
   Wstępny przydział, ponieważ maksymalne wymagania na pamięć są na ogół znane.

Anex 7.1. Algorytm bankiera *(Banker's algorithm)*

Zmodyfikowany *Graf przydziału zasobów* nie nadaje się do unikania zakleszczeń, gdy każdy typ zasobu ma wiele egzemplarzy.

Algorytm bankiera unikania zakleszczeń, działa w systemie z wieloma egzemplarzami zasobu tego samego typu.

Jest ***mniej wydajny*** od schematu grafu przydziału zasobów.

Użyty w systemie bankowym mógłby gwarantować, że bank nigdy nie zainwestuje gotówki w sposób, który uniemożliwiłby mu zaspokojenie wymagań wszystkich jego klientów.

☛ Proces **musi zadeklarować maksimum** potrzebnych egzemplarzy każdego typu zasobów.

☛ Po zamówieniu system **musi określić**, czy po ***przydzieleniu*** będzie on w stanie bezpiecznym.
Jeśli tak, to zasoby zostaną przydzielone;
w przeciwnym razie proces musi **czekać**, aż inne procesy zwolnią potrzebne zasobów.

Implementacja algorytmu zawiera zmieniające się czasie struktury danych, przechowujące stan systemu przydziału zasobów.

Niech **n** będzie liczbą procesów zaś **m** liczbą ***typów*** zasobów.

Dostepne[1..m] -liczba dostępnych zasobów każdego typu.
Dostepne[k] = r - dostępnych jest **r** egzemplarzy zasobu typu Z_k

Maksymalne[1..n, 1..m] -definiuje maksymalne żądania każdego procesu.
Maksymalne[i, k] = r, to proces P_i może zamówić co najwyżej **r** egzemplarzy zasobu typu Z_k

Przydzielone[1..n, 1..m] -liczba zasobów przydzielonych każdemu procesowi.
Przydzielone[i, k] = r -proces P_i ma przydzielonych **r** egzemplarzy zasobu typu Z_k.

Potrzebne[1..n, 1..m] -pozostałe zasoby do spełnienia zamówienia każdego z procesów.
Potrzebne[i, k] = r -do zakończenia pracy proces P_i, może potrzebować **r** dodatkowych egzemplarzy zasobu typu Z_k.

Potrzebne[i, k] = Maksymalne[i, k] - Przydzielone[i, k].

Algorytm bankiera zawiera dwa wewnętrzne algorytmy:

Algorytm bezpieczeństwa: rozstrzyga, czy system jest lub nie jest w stanie bezpiecznym.
Rząd operacji wykonywany w tym algorytmie wynosi **m** x $\mathbf{n^2}$.

Algorytm zamawiania zasobów: dopuszcza zamówienie do realizacji, jeśli stan wynikający z przydziału zasobów będzie bezpieczny.
Jeżeli nie będzie to proces musi czekać, ponadto przywrócony zostanie ***poprzedni stan*** przydziału zasobów.