Systemy operacyjne

wykład 11 - Zakleszczenia

dr Marcin Ziółkowski

Instytut Matematyki i Informatyki
Akademia im. Jana Długosza w Częstochowie

9 czerwca 2016 r.

POJĘCIE ZAKLESZCZENIA

Zakleszczenie to zbiór procesów będących w impasie wywołanym przez to, że każdy proces należący do tego zbioru przetrzymuje zasoby potrzebne innym procesom z tego zbioru, a jednocześnie czeka na zasoby przydzielone innym procesom.

PRZYKŁAD ZAKLESZCZENIA DRUKARKI

Rozważmy system, w którym są procesor i drukarka. Żeby coś wydrukować, proces musi otrzymać oba te zasoby. Przypuśćmy, że dwa procesy chcą drukować. Jednemu z nich system przydzielił drukarkę a drugiemu procesor. Pierwszy trzyma więc procesor i oczekuje na przydział drukarki, który nigdy nie nastąpi, ponieważ tę trzyma drugi proces oczekujący na przydział procesora.

PRZYKŁAD BLOKADY NA MOŚCIE

Do podobnej sytuacji może dojść na wąskim moście o ruchu dwukierunkowym. Gdy na ten most wjadą z przeciwnych kierunków dwa samochody, będą się wzajemnie blokować. Porównując to do poprzedniej sytuacji, zauważymy, że mamy do czynienia z dwoma zasobami (dwie połówki mostu). Żeby przejechać, trzeba mieć przydzielone je obydwie. W tym wypadku każdy z samochodów zajął po jednym zasobie (jednej połówce mostu) i czeka na przydział tej drugiej połówki. Znając upór polskich kierowców, można przypuszczać, że nieprędko któryś z nich ustąpi i się wycofa (dojdzie do wywłaszczenia). Inny rodzaj chamstwa drogowego nosi nazwę pociągu i polega na tym, że kierowcy z jednej strony mostu formują spójną kolumnę i bez przerwy jadą przez most, nie dając możliwości przejechania komukolwiek z drugiej strony mostu (mamy wówczas do czynienia z zagłodzeniem).

MODELOWANIE ZAKLESZCZEŃ

Rozważania o zakleszczeniach będziemy prowadzić w ramach pewnego modelu systemu. Zakłada się w nim, że w systemie jest m rodzajów zasobów oznaczanych $Z(1), Z(2), \ldots, Z(m)$. Dla każdego $i = 1, 2, \ldots, m$, $E(i)$ oznacza liczbę egzemplarzy zasobu $Z(i)$. Proces korzystający z zasobu wykonuje następujące działania:

1. żąda przydzielenia zasobu
2. korzysta z zasobu
3. zwalnia zasób

WARUNKI KONIECZNE ZAKLESZCZENIA

Aby mogło dojść do zakleszczenia muszą być spełnione warunki:

1. **Wzajemne wykluczanie** – W jednej chwili z jednego zasobu może korzystać co najwyżej jeden proces
2. **Przetrzymywanie i oczekiwanie** – Proces przetrzymujący co najmniej jeden zasób czeka na przydział dodatkowych zasobów, które są przydzielone innym procesom
3. **Brak wywłaszczania** Procesy zwalniają zasoby dobrowolnie po zakończeniu swojego zadania. Nie można odebrać procesowi przydzielonego zasobu, którego ten nie ma ochoty zwolnić
4. **Czekanie cykliczne** – Istnieje zbiór czekających procesów P(1), P(2), ..., P(n), takich, że proces P(1) czeka na zasób przydzielony procesowi P(2), proces P(2) czeka na zasób przydzielony procesowi P(3), ..., proces P(i-1) czeka na zasób przydzielony procesowi P(i), ..., proces P(n-1) czeka na zasób przydzielony procesowi P(n), proces P(n) czeka na zasób przydzielony procesowi P(1)

Oczywiście warunki te nie zapewniają, że do zakleszczenia dojdzie, jednak nie spełnienie jakiekogolwiek warunku eliminuje problem zakleszczenia.

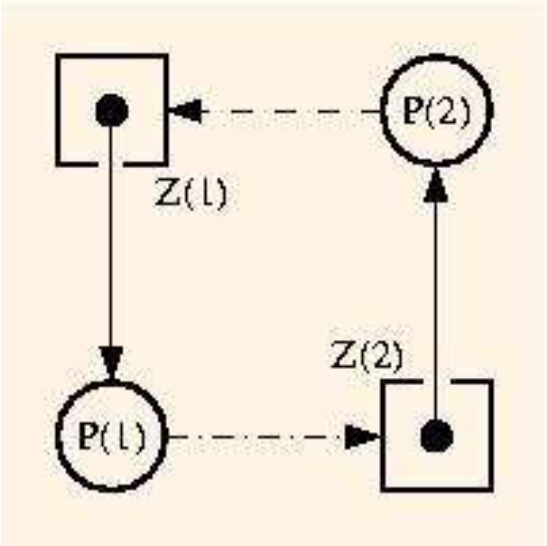
GRAF PRZYDZIAŁU ZASOBÓW

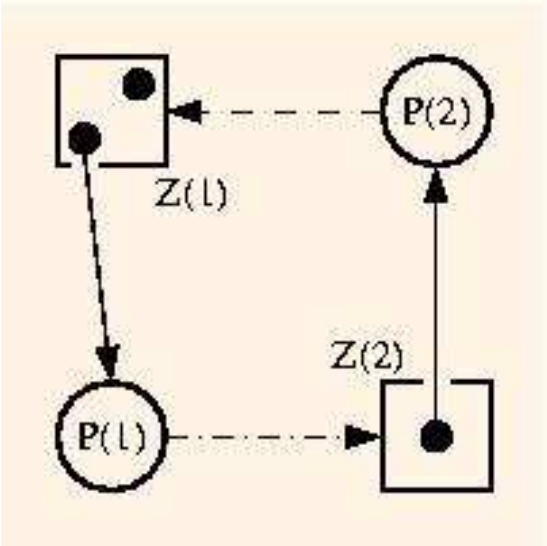
Graf przydziału zasobów jest obiektem matematycznym, za którego pomocą łatwo jest analizować zakleszczenia. Zauważmy, że jeden z warunków koniecznych zakleszczenia mówi właśnie o cyklu w jakimś grafie. Fakt istnienia, bądź nie, cyklu w tym grafie ściśle wiąże się więc z występowaniem zakleszczeń.
Graf przydziału zasobów jest skierowany.
Wierzchołkami tego grafu są procesy działające w systemie P(1), P(2), ..., P(n) oraz zasoby systemowe Z(1), Z(2), ..., Z(m).
Krawędź żądania od procesu P(i) do zasobu Z(j) oznacza, że proces P(i) zażądał przydzielenia zasobu Z(j).
Krawędź przypisania od zasobu Z(j) do procesu P(i) oznacza, że zasób Z(j) jest przydzielony procesowi P(i).

GRAF PRZYDZIAŁU ZASOBÓW

Jeśli w tym grafie występuje cykl, może to oznaczać istnienie zakleszczenia. Wyobraźmy sobie dwa procesy P(1) i P(2) oraz dwa zasoby Z(1) i Z(2), z których każdy ma tylko jeden egzemplarz. Procesowi P(1) przydzielono zasób Z(1), a procesowi P(2) przydzielono zasób Z(2). Jednocześnie proces P(1) zażądał zasobu Z(2) a proces P(2) zażądał zasobu Z(1). Te procesy są w stanie zakleszczenia. Cykl w grafie przydziału zasobów nie wystarcza do tego, żeby było zakleszczenie. Gdyby w poprzednim przykładzie były dwa egzemplarze Z(1), to oczywiście zakleszczenia by nie było. Obie sytuacje są przedstawione na rysunkach. Natomiast **jeśli w grafie przydziału zasobów jest cykl i wszystkie zasoby w tym cyklu są jednoegzemplarzowe, to na pewno ma miejsce zakleszczenie**.

INTERPRETACJA ZAKLESZCZENIA





METODY RADZENIA SOBIE Z ZAKLESZCZENIAMI

- **Zapobieganie zakleszczeniom** – polega na zaprzeczeniu co najmniej jednemu z czterech warunków koniecznych zakleszczenia. Gdy co najmniej jeden z tych warunków nie jest spełniony, mamy pewność, że do zakleszczenia nie dojdzie
- **Unikanie zakleszczeń** – nie dopuszczamy do zakleszczeń poprzez badanie stanu systemu przed każdym żądaniem przydziału zasobów i niekiedy odrzucamy to żądanie nawet wtedy, gdy są wolne zasoby, ale uznaliśmy, że spełnienie tego żądania może prowadzić do zakleszczenia. Unikanie zakleszczeń zwykle wymaga dodatkowej wiedzy o procesach (np. o ich maksymalnym dopuszczalnym zapotrzebowaniu na zasoby)
- **Wykrywanie zakleszczeń i odtwarzanie** – dopuszczamy powstawanie zakleszczeń, ale umiemy je wykrywać, likwidować i przywracać normalne działanie systemu po tym zabiegu

ZAPOBIEGANIE

Zapobieganie zakleszczeniom polega na zaprzeczeniu co najmniej jednemu z czterech warunków koniecznych zakleszczenia. Gdy co najmniej jeden z tych warunków nie jest spełniony, mamy pewność, że do zakleszczenia nie dojdzie.

Brak wzajemnego wykluczania (pierwszy warunek) oznacza eliminację całego problemu. W rzeczywistości korzystanie z większości zasobów i urządzeń wymaga wyłączności, więc zapobieganie zakleszczeniom nie może polegać na eliminacji tego warunku.

Brak przetrzymywania i oczekiwania można zrealizować na przykład w ten sposób, że proces musi zażądać kompletu potrzebnych zasobów zanim przystąpi do działania. Można też zezwolić na żądanie przydzielenia zasobów tylko tym procesom, które nie są w posiadaniu żadnego zasobu. Niestety prowadzi to do niskiego wykorzystania zasobów i możliwości zagłodzenia procesów, które jednocześnie wykorzystują wiele zasobów.

ZAPOBIEGANIE

Wywłaszczanie można zrealizować na przykład tak, że proces żądający zasobu, który nie jest obecnie dostępny, musi zwrócić wszystkie posiadane przez siebie zasoby. Następnie jest on umieszczamy w kolejce oczekiwania na zasób, którego zażądał, i zasoby, które przymusowo zwolnił. Proces ten będzie wznowiony dopiero wtedy, gdy wszystkie te zasoby będą dostępne i proces je otrzyma.

Wykluczenie czekania cyklicznego można osiągnąć poprzez ponumerowanie wszystkich rodzajów zasobów i wprowadzenie zasady, że proces musi żądać zasobów w kolejności rosnących numerów zasobów (np. nie może zażądać zasobu 4 po przydzieleniu mu zasobu 7).

UNIKANIE

Gdy stosujemy metodę unikania zakleszczeń wszystkie warunki konieczne są prawdziwe. Nie dopuszczamy do zakleszczeń poprzez badanie stanu systemu przed każdym żądaniem przydziału zasobów i niekiedy odrzucamy to żądanie nawet wtedy, gdy są wolne zasoby, ale uznaliśmy, że spełnienie tego żądania może prowadzić do zakleszczenia. Unikanie zakleszczeń zwykle wymaga dodatkowej wiedzy o procesach. Najprostszą i najbardziej przydatną informacją tego rodzaju są dane o maksymalnym dopuszczalnym zapotrzebowaniu na zasoby w każdym procesie. W algorytmie unikania dynamicznie bada się stan przed każdym żądaniem i sprawdza się, czy jego spełnienie może doprowadzić do czekania cyklicznego.

STAN BEZPIECZNY

Realizując unikanie zakleszczeń, możemy na przykład przez cały czas utrzymywać **stan bezpieczny**, tzn. taki, w którym istnieje **ciąg bezpieczny**.

CIĄG BEZPIECZNY

Ciąg procesów $P(1), P(2), \ldots, P(n)$ nazywamy bezpiecznym, gdy dla $i = 1, 2, \ldots, n$ maksymalne potrzeby procesu $P(i)$ mogą być zaspokojone za pomocą obecnie dostępnych zasobów i zasobów będących w posiadaniu procesów $P(1), P(2), \ldots, P(i-1)$.

STAN BEZPIECZNY

Intuicja kryjąca się za tą definicją polega na tym, że zakładamy, iż proces, którego maksymalne potrzeby zaspokojono, kończy się i zwalnia posiadane przez siebie zasoby. Proces $P(i)$ może więc korzystać z zasobów zwolnionych przez poprzednie procesy w ciągu, ponieważ zakładamy, że one już skończyły swoje zadania. Jeśli system jest w stanie bezpiecznym, nie ma zakleszczeń, ponieważ istnieje możliwa do zrealizowania kolejność wykonania i zakończenia wszystkich procesów. Stan niebezpieczny nie oznacza jednak konieczności istnienia zakleszczenia (np. proces nie zawsze musi żądać wszystkich zadeklarowanych przez siebie zasobów). Unikanie polega jednak na utrzymywaniu systemu w stanie bezpiecznym.

ALGORYTM BANKIERA

Algorytm bankiera służy do sprawdzenia, czy stan jest bezpieczny. Polega on na skonstruowaniu ciągu bezpiecznego. Przypuśćmy, że jest n procesów $P(1), P(2), \ldots, P(n)$ oraz m zasobów $Z(1), Z(2), ..., Z(m)$. W trakcie realizacji algorytmu bankiera wykorzystuje się następujące tablice.
Dostępne[1..m] – Dostępne[i] to liczba dostępnych w danej chwili egzemplarzy zasobu Z(i)
Max[1..n, 1..m] – Max[j, i] to maksymalna liczba egzemplarzy zasobu Z(i), których może używać proces P(j) (wg jego deklaracji)
Przydział[1..n, 1..m] – Przydział[j, i] to liczba egzemplarzy zasobu Z(i) w posiadaniu procesu P(j)

ALGORYTM BANKIERA

Potrzeby[1..n, 1..m] – Potrzeby[j, i] to liczba egzemplarzy zasobu Z(i), których dodatkowo może zażądać proces P(j)
Prawdą jest że: Potrzeby[j, i] = Max[j, i] - Przydział[j, i]
Koniec[1..n] – Koniec[j] = true wtedy i tylko wtedy, gdy proces P(j) może być zakończony w dotychczas skonstruowanym fragmencie początkowym ciągu bezpiecznego
Robocze[1..m] – Robocze[i] to liczba egzemplarzy zasobu Z(i) dostępnych po zakończeniu wszystkich procesów, dla których Koniec[j] = true. Jest to zestaw zasobów zwolnionych przez procesy znajdujące się w dotychczas skonstruowanym fragmencie początkowym ciągu bezpiecznego

ALGORYTM SPRAWDZANIA, CZY STAN JEST BEZPIECZNY

Stan systemu jest zadany przez tablice Potrzeby, Przydział i Dostępne. Zadaniem algorytmu jest stwierdzenie, czy ten stan jest bezpieczny. W algorytmie wykorzystamy tablice Koniec i Robocze. Oto kolejne kroki:
1. Zainicjuj tablicę Koniec wartościami false a tablicę Robocze zawartością tablicy Dostępne.
2. Znajdź takie j, że:
$Koniec[j] = false$
$Potrzeby[j] \leqslant Robocze$ (j to numer kolejnego procesu dodawanego na końcu już skonstruowanego fragmentu ciągu bezpiecznego; maksymalne potrzeby tego procesu mogą być zaspokojone przez zasoby zwolnione przez procesy znajdujące się wcześniej w już skonstruowanym fragmencie ciągu bezpiecznego)
3. Jeśli nie ma takiego j, idź do kroku 6.

ALGORYTM SPRAWDZANIA, CZY STAN JEST BEZPIECZNY

4. Jeśli jest takie j:
Robocze := Robocze + Przydział[j]
Koniec[j] := true
(po dodaniu procesu P(j) do konstruowanego ciągu bezpiecznego, zakładamy, że ten proces kończy się i zwalnia wszystkie swoje zasoby; są one teraz dostępne dla pozostałych procesów).
5. Wróć do kroku 2.
6. Jeśli dla każdego j = 1, 2, ..., n, Koniec[j] = true, to stan jest bezpieczny (skonstruowaliśmy ciąg bezpieczny).
7. W przeciwnym wypadku, stan nie jest bezpieczny.

OBSŁUGA ŻĄDANIA PROCESU

Przypuśćmy, że proces P(j) zażądał zasobów opisanych za pomocą tablicy Żądane[1..m] (Żądane[i] to liczba egzemplarzy zasobu Z(i), których żąda proces P(j)). Rozważmy obsługę tego żądania w systemie, w którym stosujemy algorytm bankiera przy realizacji strategii unikania zakleszczeń. Jeśli Żądane $>$ Potrzeby[j], to znaczy, że proces żąda więcej zasobów niż zadeklarowane na początku maksimum. Takie zlecenie jest odrzucane.
Jeśli Żądane $>$ Dostępne, to znaczy, że proces żąda więcej zasobów niż zadeklarowane na początku maksimum. Takie zlecenie jest odrzucane. Jeśli żaden z tych warunków nie jest spełniony, konstruujemy stan, który powstałby po spełnieniu tego żądania i sprawdzamy, czy jest on bezpieczny.

OBSŁUGA ŻĄDANIA PROCESU

Nowy stan otrzymamy wykonując następujące zmiany w tablicach:
Dostępne := Dostępne - Żądane
Przydział[j] := Przydział[j] + Żądane
Potrzeby[j] := Potrzeby[j] - Żądane
Jeśli taki stan jest bezpieczny, żądanie jest spełniane. Jeśli taki stan nie jest bezpieczny, proces P(j) musi czekać (mimo, że te zasoby są wolne! To właśnie jest koszt unikania zakleszczeń).

Stosując tę metodę radzenia sobie z zakleszczeniami, dopuszczamy powstawanie zakleszczeń, ale umiemy je wykrywać, likwidować i przywracać normalne działanie systemu po tym zabiegu.
Graf oczekiwania – Gdy w systemie występuje po dokładnie jednym egzemplarzu każdego zasobu, można skorzystać z grafu oczekiwania. Jest to uproszczona postać grafu przydziału zasobów. Graf oczekiwania jest skierowany. Wierzchołkami tego grafu są procesy działające w systemie $P(1), P(2), \ldots, P(n)$. Krawędź oczekiwania od procesu $P(k)$ do procesu $P(l)$ oznacza, że proces $P(k)$ czeka na zwolnienie zasobu przez proces $P(l)$. Istnienie cyklu w tym grafie oznacza zakleszczenie. Wykrywanie zakleszczenia może więc polegać na utrzymywaniu tego grafu i okresowe sprawdzanie istnienia w nim cyklu. Koszt takiego algorytmu (np. poprzez wyszukiwanie w głąb) jest proporcjonalny do liczby krawędzi grafu, która maksymalnie może wynieść n^2.

INNA WERSJA ALGORYTMU BANKIERA

Wykrywanie zakleszczeń można przeprowadzić też metodą podobną do algorytmu bankiera. Wykorzystuje się w nim tablice Dostępne, Przydział, Koniec i Robocze, takie jak poprzednio oraz tablicę Żądane[1..n, 1..m]. Żądane[j, i] to liczba egzemplarzy zasobu Z(i), na których przydział oczekuje proces P(j). Algorytm polega na sprawdzeniu, czy istnieje taki ciąg procesów P(1), P(2), ..., P(n), że dla i=1, 2, ..., n żądanie procesu P(i) może być zaspokojone za pomocą obecnie dostępnych zasobów i zasobów będących w posiadaniu procesów P(1), P(2), ..., P(i-1). Intuicja kryjąca się za tą definicją jest taka, jak w algorytmie bankiera. Zakładamy, iż proces, którego żądanie zaspokojono, kończy się i zwalnia posiadane przez siebie zasoby. Proces P(i) może więc korzystać z zasobów zwolnionych przez poprzednie procesy w ciągu, ponieważ zakładamy, że one już skończyły swoje zadania.

INNA WERSJA ALGORYTMU BANKIERA

1. Zainicjuj tablicę Koniec wartościami false dla procesów, które coś mają przydzielone (Przydział[j] $\neq 0$) i wartościami true dla procesów, które nic nie mają przydzielone (Przydział[j] $= 0$). Tablicę Robocze zainicjuj zawartością tablicy Dostępne.
2. Znajdź takie j, że:
Koniec[j] $=$ false
Żądane[j] $\leqslant$ Robocze (j to numer kolejnego procesu dodawanego na końcu już skonstruowanego fragmentu ciągu; żądanie tego procesu może być zaspokojone przez zasoby zwolnione przez procesy znajdujące się wcześniej w już skonstruowanym fragmencie ciągu)
3. Jeśli nie ma takiego j, idź do kroku 6.

INNA WERSJA ALGORYTMU BANKIERA

4. Jeśli jest takie j:
Robocze := Robocze + Przydział[j]
Koniec[j] := true
(po dodaniu procesu P(j) do konstruowanego ciągu, zakładamy, że ten proces kończy się i zwalnia wszystkie swoje zasoby; są one teraz dostępne dla pozostałych procesów).
5. Wróć do kroku 2.
6. Jeśli dla każdego j = 1, 2, ..., n, Koniec[j] = true, to w systemie nie ma zakleszczenia (skonstruowaliśmy scenariusz wyjścia z obecnego stanu).
7. W przeciwnym wypadku, w systemie jest zakleszczenie. Co więcej, zakleszczone są procesy, dla których Koniec[j] = false.

ODTWARZANIE

Po wykryciu zakleszczenia należy je zlikwidować. Można na przykład przerwać wszystkie zakleszczone procesy. Jest to jednak zbyt brutalna metoda. Lepiej jest przerywać zakleszczone procesy po jednym do chwili, w której zakleszczenie zniknie. Zwykle powinno wystarczyć przerwanie jednego procesu. Oto kryteria, z których można skorzystać przy wyborze procesu do przerwania:

- Priorytet procesu
- Czas działania procesu i przewidywany czas do jego ukończenia
- Zasoby przydzielone procesowi
- Zasoby potrzebne procesowi do zakończenia działania
- Ile procesów trzeba będzie zakończyć?
- Czy proces jest wsadowy, czy interakcyjny?

ODTWARZANIE

Przy wyborze ofiary należy minimalizować koszt takiej decyzji, np. nie zabijać procesu, który zaraz się skończy, zabijać jak najmniej procesów etc. Jeśli to możliwe, zamiast przerywać proces można go wycofać do jakiegoś wcześniejszego bezpiecznego stanu (tak robi się w bazach danych przy zerwaniu transakcji!). Należy też zwracać uwagę na zagłodzenie – pewien proces może zawsze stawać się ofiarą odtwarzania po zakleszczeniu i nigdy nie zakończyć działania. Rozwiązaniem może być tu dynamiczne zwiększanie priorytetu procesu po każdym zakleszczeniu, którego był ofiarą.