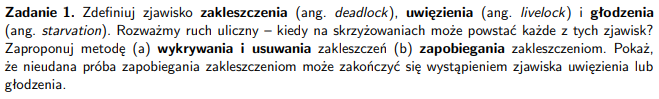
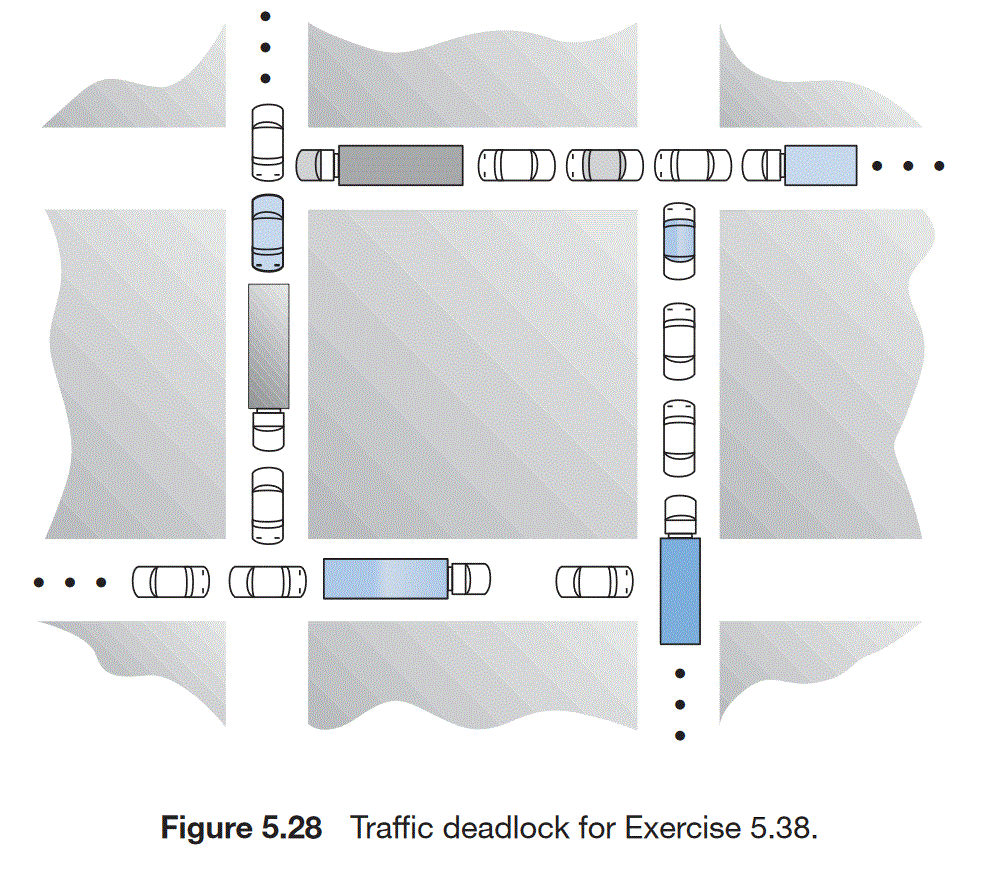
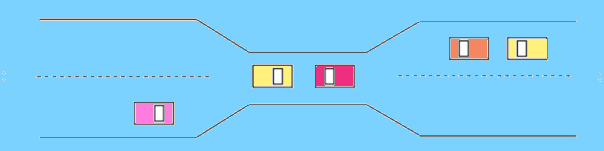
Systemy Komputerowe – Lista C



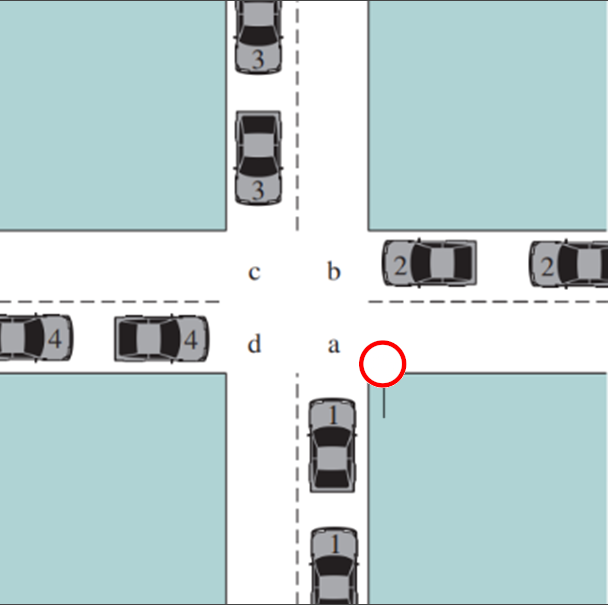
**Zakleszczenie (deadlock)** - Mamy kilka procesów, gdzie każdy z procesów czeka na zakończenie akcji wykonywanej przez inny: P1 czeka na P2, P2 czeka na P3, P3 czeka na P1. Więc mamy cykl procesów czekających na siebie.



**Uwięzienie (livelock)** – Podobne do zakleszczenia, ale tutaj każdy z procesów zmienia cały czas swój stan w zależności od innych, nie wykonując żadnych akcji. P1 potrzebuje zasobów R1 i R2, tak samo P2. P1 blokuje zasób R1 i czeka na zasób R2, który jest zablokowany przez P2. P1 oddaje zasób R1, tak samo P2 zasób R2. Jednak w tym samym czasie próbują ponownie zablokować zasoby, więc znowu P1 ma jeden zasób, P2 drugi i nie mogą przetworzyć danych.



**Głodzenie (starvation)** – Proces jest głodzony jeśli z jakiegoś powodu nie dostaje dostępu do zasobu, którego potrzebuje do działania



Wykrywanie i usuwanie zakleszczeń – Zakleszczenia występują, ale w momencie ich wystąpienia system stara się to wykryć i następnie rozwiązać problem.

-Wykrywanie zakleszczenia:

-Zasoby pojedynczej instancji

Możemy stworzyć graf skierowany alokacji zasobów. Rozpisujemy jaki proces potrzebuje jakich zasobów. Jeżeli w takim grafie wykryjemy cykl, wtedy mamy zakleszczenie.

-Zasoby mnogiej instancji

Tzw. „Banker’s algorithm”. Mamy macierze „Allocation”, „Request” oraz wektor “Available”.

Allocation – Jak dużo instancji danego zasobu trzyma dany process.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | A | B | C | D |
| P0 | 0 | 0 | 1 | 2 |
| P1 | 1 | 0 | 0 | 0 |
| P2 | 1 | 3 | 5 | 4 |
| P3 | 0 | 6 | 3 | 2 |
| P4 | 0 | 0 | 1 | 4 |

P0 zaalokował 0 instancję zasobu A, 0 instancje zasobu B, 1 zasobu C itd. Analogicznie pozostałe procesy.

Request – Jak dużo instancji danego zasobu może sobie zażyczyć dany proces.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | A | B | C | D |
| P0 | 0 | 0 | 1 | 2 |
| P1 | 1 | 7 | 5 | 0 |
| P2 | 2 | 3 | 5 | 6 |
| P3 | 0 | 6 | 5 | 2 |
| P4 | 0 | 6 | 5 | 6 |

P0 może sobie zażyczyć maksymalnie 0 instancji zasobu A, 0 zasobu B, 1 zasobu C itd. Analogicznie pozostałe.

Available – Jak dużo instancji danego zasobu jest dostępnych w danej chwili.

|  |  |  |  |
| --- | --- | --- | --- |
| A | B | C | D |
| 1 | 5 | 2 | 0 |

Możemy uzyskać dostęp do 1 instancji zasobu A, 5 zasobu B, 2 zasobu C itd.

Tworzymy macierz „need”. Czyli różnicę macierzy Request i Allocation. To da nam informację ile potrzebujemy instancji danego zasobu dla danego procesu, aby go wykonać. Następnie przechodzimy po kolei po wierszach tej macierzy i sprawdzamy czy przy pomocy wektora „Available” jesteśmy w stanie wyzerować elementy danego wiersza. Jeśli tak, zaznaczamy wiersz.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | A | B | C | D |
| P0 | 0 | 0 | 0 | 0 |
| P1 | 0 | 7 | 5 | 0 |
| P2 | 1 | 0 | 0 | 2 |
| P3 | 0 | 0 | 2 | 0 |
| P4 | 0 | 6 | 4 | 2 |

W przykładzie możemy wykonać teraz proces P0 lub P3. Po wykonaniu procesu P0 jego zasoby zostaną zwolnione, więc będziemy mieli więcej zasobów do wykorzystania w wektorze Available (zwolnione zasoby wskoczą do tego wektora). Tak postępujemy ze wszystkimi procesami. Jeśli w danym przejściu nie oznaczymy żadnego procesu, procesy, które zostały nieoznaczone są zakleszczone.

**Usuwanie zakleszczenia:**  
- Wywłaszczyć proces i tymczasowo zwolnić zasoby, których używa. Zwiększy nam to pulę wolnych zasobów do wykorzystania dla procesów.

- Zapisanie stanu procesu co jakiś czas jako „checkpointy”. Kiedy wykryjemy deadlock, cofamy proces (ten, który blokuje zasób) do ostatniego checkpointa.

- Zabicie jednego z procesów, które trzymają zasób.

**Zapobieganie zakleszczeniom** – Dążymy do tego, aby zakleszczenia nie mogły wystąpić.

- Unikamy przypisywania zasobów procesom (chyba, że to niezbędne)

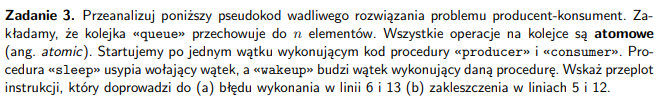
- Eliminujemy sytuację „trzymania” zasobów przez proces. Np. przed rozpoczęciem wykonania proces musi określić jakich zasobów będzie potrzebował (nie zawsze możliwe) lub jeśli proces czeka na zasób inny, zwolnienie jego dotychczas zaalokowanych zasobów (przypisanie ich gdzie indziej).

- Eliminujemy możliwość wystąpienia cyklu alokacji. Proces może uzyskać dostęp tylko do jednego zasobu na raz. Jeżeli potrzebuje kolejnego, zwalnia poprzedni.

Nieudana próba zapobiegania może doprowadzić do:

-Uwięzienia – Kilka zasobów zwalnia wszystkie swoje zasoby kiedy nie może zaalokować sobie wszystkich, przez co kilka procesów spróbuje się nawzajem „przepuścić” w kolejce. W efekcie żaden z nich się nie wykona, bo nie będzie miał wszystkich zasobów do wykonania.

-Głodzenia – Proces może nigdy się nie rozpocząć, bo nie ma zasobów.

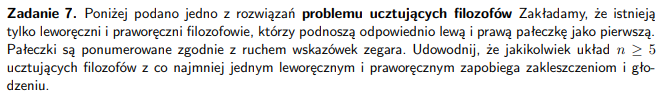


1. Błąd wykonania w linii 6:
   1. 14: konsument sprawdza, że kolejka nie jest pełna
   2. producent zapełnia kolejkę, widzi, że jest pełna i zasypia
   3. 15: konsument wybudza producenta
   4. 6: producent próbuje dodać coś do pełnej kolejki

Błąd wykonania w linii 13:

* 1. 7: producent sprawdza, że kolejka nie jest pusta
  2. konsument opróżnia kolejkę, sprawdza, że jest pusta i zasypia
  3. 8: producent wybudza konsumenta
  4. 13: konsument próbuje usunąć coś z pustej kolejki

1. Zakleszczenie w liniach 5 i 12:
   1. 11: konsument sprawdza, że kolejka jest pusta
   2. producent wypełnia kolejkę elementami do pełna i dociera do ifa sprawdzającego czy kolejka jest pełna (jest, bo właśnie ją wypełnił)
   3. 12: konsument zasypia
   4. 5: producent zasypia



**Problem ucztujących filozofów** – Wokół okrągłego stołu siedzą filozofowie nad miskami z jedzeniem. Filozofowie na zmianę myślą i jedzą. Każdy z nich potrzebuje dwóch pałeczek, aby zacząć jeść. Pałeczki są rozłożone w miejscach pomiędzy filozofami na stole. Każdy z filozofów może wziąć tylko pałeczkę, która leży obok niego (w zależności czy jest prawo, czy lewo ręczny bierze odpowiednio najpierw prawą, potem lewą pałeczkę). Jeśli nie ma możliwości jedzenia, myśli (czeka). Jeśli dostaje dwie pałeczki – je (pracuje). Po czasie każdy filozof odkłada pałeczki i proces się powtarza.

1. Zapobieganie zakleszczeniom

Musimy pokazać, że zawsze będzie taki filozof, który może jeść (a skoro on może jeść, to stan układu może się zmienić, więc zakleszczenie zniknie). Załóżmy zatem, że doszło do zakleszczenia, czyli mamy sytuację czekania na pałeczkę. Mamy takie możliwe sytuacje:

a) L | L – Pałeczkę podnosi filozof prawy (gdyby wziął lewy – mógłby już jeść – nie ma zakleszczenia)

b) P | L – Pałeczkę bierze filozof lewy lub prawy (wszystko jedno który, to bez znaczenia) – czyli pałeczki na stole nie ma

c) P | P – Pałeczkę bierze lewy filozof – Pałeczki na stole nie ma (gdyby wziął prawy, mógłby jeść – nie ma zakleszczenia)

d) L | P – Pałeczkę może wziąć filozof lewy, jeśli wziął już pałeczkę lewą, lub prawy jeśli wziął już pałeczkę prawą (tutaj mogło powstać zakleszczenie)

Czyli nieużywana pałeczka może istnieć tylko pomiędzy leworęcznym a praworęcznym filozofem (jeśli żaden z nich nie ma pałeczki – bo obaj czekają na odpowiednią dla siebie pałeczkę – lewy na lewą, prawy na prawą). Jeśli oznaczymy ilość nieużywanych pałeczek jako X, to mamy 2\*X filozofów bez pałeczek (na każdą pałeczkę przypada 2 filozofów, przekrój par filozofów jest pusty, ponieważ jeśli weźmiemy pod uwagę środkową parę (wytłuszczona):

L | P | **L | P** | L | P

to filozof obok filozofa z rozpatrywanej pary nie jest zakleszczony (z lewej strony mamy sytuację P | L, więc bez zakleszczenia, z prawej także P | L, więc także brak zakleszczenia). Więc pary filozofów są rozłączne.

Wiemy z założenia, że X jest dodatnie (istnieje taka pałeczka, która nie została podniesiona – mamy wtedy zakleszczenie).

Mamy:

X – leżące pałeczki  
N-X – Podniesione pałeczki  
2\*X – Ilość filozofów, którzy nie mają żadnej pałeczki  
N – (2\*X) – Ilość filozofów, którzy mają co najmniej jedną pałeczkę

Więc (używane pałeczki) = N-X > N – (2\*X) = (jedzący filozofowie) => Z zasady szufladkowej widzimy, że istnieje filozof, który ma dwie pałeczki (je), więc nie ma zakleszczenia, bo stan układu może się zmienić – SPRZECZNOŚĆ.

1. Zapobieganie zagłodzeniu

Musimy pokazać, że każdy filozof będzie miał okazję w którymś momencie coś zjeść (nie będziemy go głodzić). Weźmy dowolnego filozofa X, który nie może w tym momencie zjeść.

1. Jeśli X jest praworęczny

Według poprzedniego zadania, możemy mieć tylko sytuację, że filozof praworęczny ma jedną łyżkę w ręce (a innym przypadku wszystko byłoby rozwiązane). Więc filozof X ma w ręce pałeczkę (w prawej ręce). Czeka na pałeczkę lewą. Sąsiad X po lewej to Y.

Jeśli Y w danym momencie je, to kiedy skończy – X będzie miał okazję wziąć jego pałeczkę. Jeśli Y nie je:

- Y jest leworęczny  
Jeśli nie je i jest leworęczny, to nie ma swojej lewej pałeczki (według poprzedniego podpunktu) – X bierze jego pałeczkę prawą i właśnie je.

- Y jest praworęczny  
Y trzyma swoją prawą pałeczkę i czeka na lewą. Weźmy Z jako pierwszego leworęcznego filozofa na lewo od Y. Jeśli Z właśnie je, to po skończeniu jego prawy sąsiad weźmie pałeczkę, po skończeniu jedzenia sąsiad prawy tego sąsiada weźmie pałeczkę itd, aż pałeczkę będzie mógł wziąć Y. Jeśli Z nie je, to czeka na swoją lewą pałeczkę, więc prawa pałeczka jest wolna, więc po kolei filozofowie na prawo będą mogli jeść aż przyjdzie czas na Y.

b) Jeśli X jest leworęczny – Analogicznie, zmienione kierunki