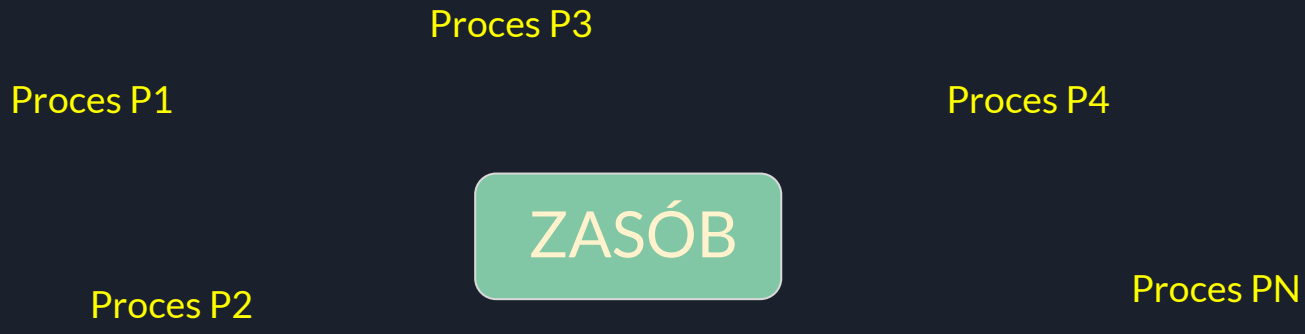
Rozproszona sekcja krytyczna

Wymagania do projektów zaliczeniowych
oraz przykładowe algorytmy

O czym powiemy?

- Czym jest rozproszona sekcja krytyczna?
- Jak trywialnie prosto rozwiązać?
- Wymagania co do projektów
- Naiwny, błędny algorytm
- Algorytm Lamporta
- Algorytm Ricarta-Agrawali

Rozproszona sekcja krytyczna



N procesów ubiegających się o dany zasób (względnie: w uogólnionej sekcji krytycznej, o *N* zasobów) **w środowisku rozproszonym**.

Tematy zaliczeniowe

Otrzymacie do rozwiązania problemy dostępu do rozproszonej sekcji krytycznej. Zaprojektujecie w pełni rozproszone algorytmy (wymagania na kolejnych slajdach) w grupach dwuosobowych.

Najpierw mają zostać przygotowane algorytmy; dopiero po ich akceptacji można przystąpić do implementacji.

Projekty można oddawać do ostatniego tygodnia zajęć. Następnie, każde dwa tygodnia opóźnienia skutkują obniżeniem oceny o pół stopnia.

Rozwiązanie trywialne

- Proces P jest desygnowany jako koordynator-zarządca.
- Pozostałe procesy zgłaszają zapotrzebowanie do zarządcy.
- Zarządca przydziela zasoby w kolejności zgłoszenia (utrzymując kolejkę zgłoszeń).
- Po wykorzystaniu zasobu proces wysyła komunikat o zwolnieniu do zarządcy: zarządca usuwa proces z kolejki i przesyła zezwolenie na wykorzystanie zasobu do innego procesu.

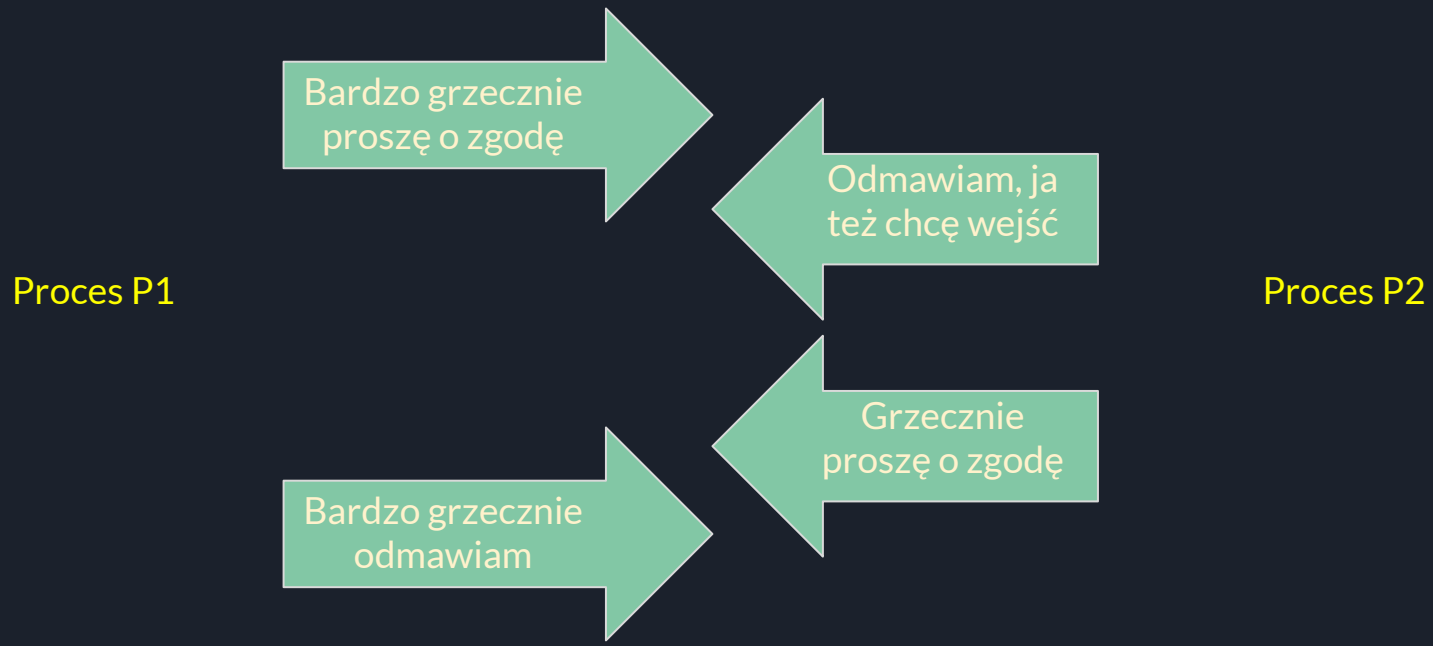
A czy dałoby się rozwiązać problem w sposób **rozproszony**, **bez centralnego zarządcy**? Oczywiście!

Naiwny, błędny algorytm

1. Process prosi pozostałe procesy o zgodę na wejście do sekcji krytycznej.
2. Po otrzymaniu prośby o zgodę proces zgadza się, o ile sam się nie ubiega oraz o ile dotąd nie udzielił jeszcze zgody.
3. Po zebraniu wszystkich zgód, proces uzyskuje dostęp do sekcji krytycznej (do zasobu).

Nikt się nie dostanie do sekcji!

Problem z naiwnym algorytmem

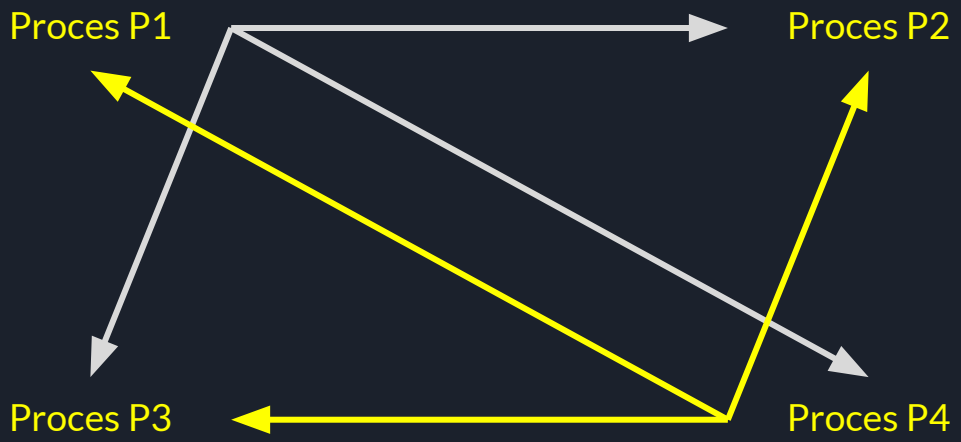


Naiwny, błędny algorytm (2)

1. Process prosi pozostałe procesy o zgodę na wejście do sekcji krytycznej.
2. Po otrzymaniu prośby o zgodę proces zgadza się, o ile sam się nie ubiega oraz o ile dotąd nie udzielił jeszcze zgody; Jeżeli ubiega się, zgadza się, o ile ma wyższy priorytet
3. Po zebraniu wszystkich zgód, proces uzyskuje dostęp do sekcji krytycznej (do zasobu).

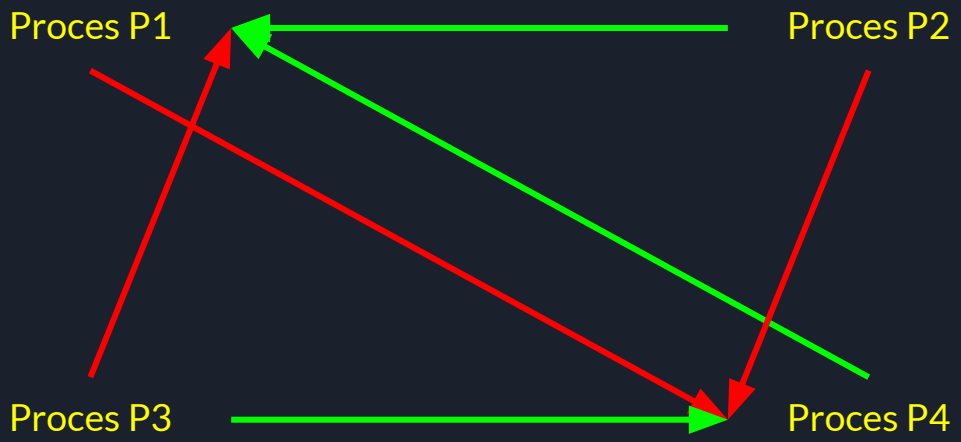
Zakleszczenie!

Zakleszczenie (1) - Prośby.



Procesy P1 i P4 chcą się dostać do sekcji krytycznej

Zakleszczenie (2) - Odpowiedzi



Zielony: zgoda.
Czerwony: brak odpowiedzi lub brak zgody

Algorytm Lamporta

Każde żądanie procesu *i*-tego otrzymuje priorytet. Najprościej: zegar Lamporta z czasu żądania. Ew. konflikty rozstrzygamy po identyfikatorach. Ważne: priorytety ZAWSZE muszą pozwalać na jednoznaczne rozstrzyganie konfliktów!

Procesy utrzymują lokalną kolejkę żądań, posortowaną po priorytetach.

Musimy wiedzieć, że nasze żądanie ma najwyższy priorytet w systemie. Dlatego dla każdego innego procesu musimy wiedzieć, że albo też żąda, ale ma mniejszy priorytet; albo nie żąda, ale *gdyby żądał*, to jego priorytet jest mniejszy.

Pamiętamy, że zegary Lamporta są rosnące! Czyli jak dostaniemy od procesu *j* wiadomość nie będącą żądaniem z timestampem *k*, to ewentualne żądanie musi być co najmniej *k*+1

Co oznacza, że musimy założyć kanały FIFO...

Algorytm Lamporta

Aby otrzymać dostęp do sekcji, proces *i*-ty:

1. Wysyła żądanie REQ do wszystkich procesów oraz wstawia swoje żądanie do lokalnej kolejki.
2. Proces otrzymuje dostęp do sekcji, gdy zachodzą równocześnie dwa warunki:
   a. żądanie procesu jest na szczycie lokalnej kolejki
   b. od każdego innego procesu otrzymaliśmy wiadomość o wyższym zegarze Lamporta od żądania procesu *i*-tego
3. Po wyjściu z sekcji krytycznej wysyłamy RELEASE do wszystkich

Algorytm Lamporta

1. Przy otrzymaniu żądania REQ od innego *j*-tego procesu, proces *i*-ty wstawia je do lokalnej kolejki.
2. Aby zapewnić, że proces *j*-ty na pewno otrzyma od nas wiadomość ze starszym zegarem Lamporta, wysyłamy mu potwierdzenie ACK
3. Po otrzymaniu wiadomość RELEASE usuwamy żądanie procesu z lokalnej kolejki

Złożoność komunikacyjna i czasowa

Aby *n*-1 REQ, *n*-1 ACK oraz *n*-1 RELEASE: razem przesyłamy się 3n-3 wiadomości. Jeżeli przyjmiemy, że wiadomości wysyłamy też sami do siebie, to złożoność komunikacyjna wynosi 3n

Trzy rundy: w pierwszej rundzie równocześnie do wszystkich idą wiadomość REQ. Wszyscy otrzymują je w tej samej chwili i w tej samej chwili odsyłają ACK, które docierają do nas w tej samej chwili. Potem pozostaje RELEASE do wszystkich. Pomijając więc czasy przetwarzania lokalnego oraz przyjmując jednostkowy koszt przesłania wiadomości, złożoność czasowa wynosi więc 3.

Algorytm Ricarta-Agrawali

W sumie, po co przesyłać ACK, skoro proces i tak musi na nie czekać i dopóki ich nie otrzyma, nie może podjąć decyzji?

Proces *i*-ty, aby dostać się do sekcji krytycznej:

1. Wysyła REQ do wszystkich
2. Po zebraniu ACK od wszystkich, może wejść do sekcji krytycznej.

Algorytm Ricarta-Agrawali

Proces *i*-ty po otrzymaniu REQ od procesu *j*-tego:

1. Jeżeli nie ubiega się o sekcję, wysyła ACK (pamiętamy o aktualizacji własnego zegara Lamporta w zwykły sposób!)
2. Jeżeli ubiega się o sekcję i ma gorszy priorytet, wysyła ACK
3. Wpw, wstawiamy REQ procesu *j*-tego do kolejki oczekujących

Po wyjściu z sekcji krytycznej proces *i*-ty sprawdza, czy kolejka oczekujących jest pusta. Jeżeli jest niepusta, wysyła ACK do wszystkich oczekujących procesów. Jeden z nich (i tylko dokładnie jeden!) będzie miał teraz komplet ACKów i będzie mógł wejść do sekcji krytycznej.

Złożoność komunikacyjna i czasowa

1. Złożoność komunikacyjna: 2n
2. Złożoność czasowa: 2

Optymalizacja?

Jeżeli proces *i*-ty wysłał żądanie z priorytetem *k* do procesu *j*-tego, i otrzymał żądanie z priorytetem *a*, *a*<*k* od procesu *j*-tego… to wie, że proces *j* musi wysłać ACK. A w takim razie nie musi na ten ACK czekać. Z drugiej strony, proces *j* będzie wiedział, że proces *i*-ty będzie wiedział, że *j* musi wysłać ACK i nie będzie na ten ACK czekał. A w takim razie: to po co ten ACK wysyłać?

Oszczędzamy więc jedną wiadomość ACK. W sytuacji, gdy wszyscy równocześnie się ubiegają o sekcję, pierwszy proces dostaje się do sekcji krytycznej w pierwszej rundzie i zaoszczędzamy *n* ACKów!

Wymagania co do projektów zaliczeniowych (1)

- Rozwiązania w pełni rozproszone (brak elementów centralnych)
- Brak globalnych zamków (w których tylko jeden proces naraz może podjąć decyzję, a reszta czeka na swoją kolej)
- Mniej rund == lepiej (optymalizacja złożoności czasowej), w drugiej kolejności mniej komunikatów == lepiej (optymalizacja złożoności komunikacyjnej). Prowadzący może zgodzić się na inne optymalizacje lub inny nacisk na złożoność

Wymagania (2)

Procesy muszą wypisywać, co w danej chwili robią: podstawowe komunikaty typu “chcę wejść do sekcji krytycznej” “jestem w sekcji” “wychodzę” “jestem poza sekcją”

Wykorzystujemy PVM/MPI, dowolny język programowania, tylko C/C++ ma gwarancję wsparcia. Dla pozostałych należy samodzielnie poprosić administratora laboratoriów o doinstalowanie potrzebnych bibliotek

Kod ma być czytelny, dobrze opisany.

Dołączamy sprawozdanie, wydrukowane na papierze i podpisane długopisem, z opisem problemu, opisem algorytmu (najlepiej w postaci maszyny stanów) oraz złożonością komunikacyjną i czasową