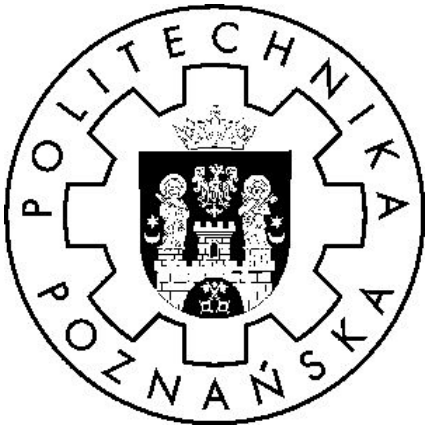

Rozproszona sekcja krytyczna

Przetwarzanie Rozproszone, lab. nr 5A

Czym jest rozproszona sekcja krytyczna?

Jeżeli w środowisku rozproszonym mamy *n* procesów ubiegających się o dostęp do zasobu, z którego tylko jeden naraz może skorzystać, mamy do czynienia z rozproszoną sekcją krytyczną.

Jeżeli zasób równocześnie może używać *k* procesów, mamy do czynienia z uogólnioną sekcją o pojemności *k*.

Przykład: mamy osiedle domków jednorodzinnych w czasie pandemii oraz sklepik z żywnością. Tylko jedna osoba naraz może być w sklepiku. Osoby więc w jakiś sposób przy pomocy komórek i SMSów ustalają, kto może w danej chwili wejść do sklepiku.

Jeżeli w sklepiku naraz mogą się znaleźć trzy osoby, dalej mamy sekcję krytyczną, tyle, że o pojemności trzy.

Rozwiązanie trywialne

Najprościej rozwiązać problem w sposób scentralizowany. Desygnujemy jeden proces, by działał jako zarządca. Zarządca posiada kolejkę żądań. Każdy proces wysyła żądanie do zarządcy. Zarządca kolejkuje żądania wg kolejności zgłoszeń i wysyła zgodę jednemu procesowi naraz. Proces wychodząc z sekcji krytycznej powiadamia zarządcę, który może go usunąć z kolejki i wybrać kolejny proces, któremu może wysłać zezwolenie.

Oczywiście, na przedmiocie o nazwie *Przetwarzanie Rozproszone* nie możemy patrzeć na takie rozwiązanie bez obrzydzenia. Rozwiązania scentralizowane posiadają mnóstwo zalet oraz sporo wad. Nam jednak zależy (nam, to znaczy prowadzącym i wam, studentom. Nie macie wyjścia, musi wam zależeć), by spróbować stworzyć algorytm obywający się bez centralnego zarządcy

Naiwne i błędne rozwiązanie

Pierwszym podejściem do rozwiązania może być wymóg, by proces prosił pozostałe o zgodę. Dla ułatwienia, rozważmy tylko jedną rundę (tj. wybieramy tylko jeden proces z wielu ubiegających się i nie dbamy, co się dzieje dalej). W takim wypadku opis algorytmu wygląda następująco:

Używamy dwóch typów wiadomości:

REQ - prośby o dostęp do sekcji

ACK - udzielenie zgody na dostęp do sekcji

Proces *i*-ty ubiegający się o dostęp do sekcji postępuje następująco:

1. Proces wysyła **REQ** do wszystkich procesów
2. Proces dostaje się do sekcji po zebraniu **ACK** od wszystkich procesów

W każdej chwili, proces następująco reaguje na wiadomości:

Po otrzymaniu **REQ**:

Jeżeli proces nie ubiega się o dostęp do sekcji, wysyła **ACK**

Jeżeli proces ubiega się o dostęp do sekcji, nic nie wysyła

Po otrzymaniu **ACK**:

Proces zwiększa licznik otrzymanych **ACK**ów i wchodzi do sekcji, jeżeli otrzyma **ACK** od wszystkich procesów.

Widać oczywiście na pierwszy rzut oka, że taki algorytm nie spełnia warunku postępu: jeżeli dwa procesy równocześnie ubiegają się o wejście do sekcji, żaden z nich nie wyśle drugiemu zgody i żaden z nich nie otrzyma dostęp.

Zmodyfikujmy więc algorytm następująco:

Żądania procesów mają przydzielony priorytet.

Po otrzymaniu **REQ**:

Jeżeli proces nie ubiega się o dostęp do sekcji, wysyła **ACK**

Jeżeli proces ubiega się o dostęp do sekcji i ma niższy priorytet, wysyła **ACK**;

Jeżeli proces ubiega się o dostęp do sekcji i ma wyższy priorytet, nic nie wysyła.

Również ta wersja oczywiście nie spełnia warunku postępu. Załóżmy trzy procesy, P1, P2, P3. P1 oraz P2 ubiegają się o wejście do sekcji krytycznej, P1 ma wyższy priorytet od P2.

Mamy następującą sytuację:

P1 wysyła REQ1 do P2 oraz P3

P2 wysyła REQ2 do P1 oraz P3

odbiera od i odpowiada ACK

P3 odbiera REQ2 od P2 i odpowiada ACK

P3 odbiera REQ1 od P1 i nie odpowiada (bo już wysłał ACK do P2)

P1 odbiera REQ2 od P2 i nie odpowiada (bo ma wyższy priorytet).

Żaden proces nie zbierze wymaganej liczby zgód.

Widać jednak chyba, że ten algorytm można zmodyfikować dalej, by uzyskać działające rozwiązanie problemu. Do sekcji krytycznej uzyska dostęp ten proces, który aktualnie ma najwyższy priorytet. Pozostały dwie kwestie do rozwiązania: jak dobrać priorytet, tak by jednoznacznie rozwiązywać konflikty (jak między P1 a P2 w powyższym przykładzie), oraz jak zmodyfikować algorytm, by działał więcej rund niż jedną.

Algorytm Lamporta

Jako priorytet żądań ustalimy zegar Lamporta. Priorytet nadajemy żądaniom, nie procesom; zapewni to sprawiedliwy dostęp do zasobów. Zegary lokalne modyfikujemy w normalny sposób. Gdy generujemy żądanie, przypisujemy mu aktualną wartość lokalnego zegara Lamporta.

Dostęp do sekcji ma proces o żądaniu z najwyższym priorytetem. Ponieważ dysponujemy tylko wiedzą lokalną, musimy wiedzieć dla każdego innego procesu:

> Czy inny proces się ubiega, a jeżeli tak, musimy wiedzieć, jaki priorytet ma jego żądanie
>
> Jeżeli inny proces się nie ubiega, musimy znać jego zegar Lamporta by wiedzieć, czy *gdyby się jednak zaczął ubiegać*, czy jego żądanie byłoby lepsze od naszego

Każdy proces utrzymuje lokalną kolejkę żądań, posortowaną po ich znacznikach czasowych (priorytetach żądań). Dodatkowo, każdy proces pamięta *n*-elementowy wektor zawierający zegar Lamporta ostatnio otrzymanej wiadomości od wszystkich innych procesów.

Mamy trzy typy wiadomości:

REQ - żądania dostępu

RELEASE - zwolnienie zasobu

ACK - potwierdzenie, że wiemy o cudzym żądaniu.

Aby otrzymać dostęp do sekcji krytycznej, proces *i*-ty p-ty ostępuje następująco:

> Proces wysyła **REQ** do wszystkich innych procesów łącznie z samym sobą (co powoduje wstawienie własnego żądania do kolejki żądań zgodnie z punktem poniżej przy opisie obsługi **REQ**)

Proces otrzymuje dostęp do sekcji krytycznej, gdy zachodzą równocześnie dwa warunki:

> (W1) Własne żądanie jest na szczycie kolejki żądań
>
> (W2) Od wszystkich pozostałych procesów otrzymaliśmy wiadomości o starszej etykiecie czasowej

Obsługa wiadomości przez proces *i*-ty:

> Aktualizujemy zegar Lamporta w zwykły sposób, oraz *j*-tą pozycję wektora znaczników czasowych ostatnio otrzymanych wiadomości (czyli od procesu *j*-tego)
>
> Po otrzymaniu **REQ** od procesu *j*-tego, proces *i*-ty wstawia żądanie do lokalnej kolejki żądań (posortowanej po ich znacznikach czasowych/priorytetach). Następnie wysyłamy **ACK**. Zgodnie z obsługą zegarów Lamporta, znacznik **ACK** musi być większy niż znacznik otrzymanego **REQ**.
>
> > **UWAGA:** jeżeli proces *i*-ty jeszcze się nie ubiega, to gdyby zaczął się ubiegać w przyszłości o dostęp do sekcji krytycznej, to priorytet/znacznik czasowy przypisany do tego hipotetycznego żądania w przyszłości będzie musiał być większy od wysłanego *teraz* **ACK**.
>
> Po otrzymaniu **RELEASE** od procesu *j*-tego, usuwamy jego żądanie z kolejki.
>
> **ACK** powoduje tylko aktualizację zegara lokalnego oraz wektora znaczników czasowych ostatnio otrzymanych wiadomości, jak to już opisano wyżej.

Po obsłudze wiadomości sprawdzamy, czy obecnie zachodzą warunki (W1) oraz (W2). Jeżeli tak, wchodzimy do sekcji krytycznej. Po wyjściu z sekcji proces *i*-ty wysyła **RELEASE** do wszystkich innych procesów.

Złożoność czasowa wynosi trzy (trzy rundy) przy założeniu pomijalnych czasów przetwarzania lokalnego oraz jednostkowych czasów przesłania wiadomości, zaś komunikacyjna 3n (3n-3, gdybyśmy nie wysyłali wiadomości sami do siebie).

Algorytm Ricarta-Agrawali

Łatwo zauważyć, że procesy w poprzednim algorytmie musiały czekać na wiadomość typu **ACK**. Moglibyśmy więc wrócić do naszego naiwnego algorytmu z początku dokumentu i zaproponować następujące rozwiązanie:

Każde żądanie ma przypisany priorytet - najlepiej zegar Lamporta, ale to może być dowolny priorytet, byle jednoznacznie pozwalał na rozstrzyganie konfliktów. W przypadku, gdy dwa żądania mają identyczny priorytet, decyduje identyfikator procesu-właściciela żądania.

Procesy utrzymują zegary Lamporta w zwykły sposób. W chwili utworzenia żądania, żądanie otrzymuje bieżącą wartość zegara Lamporta jako priorytet. Oczywiście, wszystkie wiadomości muszą mieć przypisany znacznik czasowy, tak, jak to wynika z algorytmu utrzymania zegara Lamporta.

Używamy dwóch typów wiadomości: **REQ** (prośba o dostęp) oraz **ACK** (udzielona zgoda)

Procesu utrzymują kolejkę oczekujących na wiadomość typu **ACK** oraz licznik zgód**.**

Aby otrzymać dostęp do sekcji krytycznej, proces *i*-ty p-ty postępuje następująco:

> Proces wysyła **REQ** do wszystkich innych procesów, ustawia licznik zgód na zero.

Proces otrzymuje dostęp do sekcji krytycznej, gdy zachodzą równocześnie dwa warunki:

> (W1) Proces otrzymał wiadomości **ACK** od wszystkich innych procesów

Obsługa wiadomości przez proces *i*-ty:

> Aktualizujemy zegar Lamporta w zwykły sposób
>
> Po otrzymaniu **REQ** od procesu *j*-tego,
>
> > Jeżeli proces *i*-ty nie ubiega się o sekcję, wysyła **ACK**. UWAGA: jeżeli w przyszłości proces *i*-ty zmieni zdanie i zacznie się ubiegać, jego **REQ** będzie miało większy zegar Lamporta niż obecnie wysłany **ACK**.
> >
> > Jeżeli proces *i*-ty ubiega się o sekcję ale jego żądanie ma niższy priorytet, wysyła **ACK**.
> >
> > Jeżeli proces *i*-ty ubiega się o sekcję ale jego żądanie ma wyższy priorytet, nie wysyła **ACK**, ale zapamiętuje proces *j*-ty w kolejce procesów oczekujących
>
> Po otrzymaniu **ACK** proces *i*-ty zwiększa licznik zgód. Jeżeli otrzymał zgody od wszystkich innych procesów, wchodzi do sekcji krytycznej. Po wyjściu z sekcji wysyła **ACK** do wszystkich procesów w kolejce procesów oczekujących.

Złożoność komunikacyjna pakietowa wynosi 2n (a dokładniej, 2n-2), zaś czasowa: 2.

Łatwo zauważyć, że algorytm można dalej zoptymalizować. Jeżeli proces *i*-ty otrzyma **REQ** z priorytetem powiedzmy 5 od procesu *j*-tego, a sam wcześniej wysłał do procesu *j*-tego **REQ** z priorytetem powiedzmy 4, to proces *i*-ty może wywnioskować:

> Proces *j*-ty ma niższy priorytet, a więc na pewno musi wysłać **ACK**. Można więc zachowywać się tak, jakby już wysłał, a więc - nie czekać na **ACK** od procesu *j*-tego i od razu zwiększyć licznik zgód.

Z drugiej strony, jeżeli proces *i*-ty otrzyma **REQ** z priorytetem powiedzmy 5 od procesu *j*-tego, a sam wcześniej wysłał do procesu *j*-tego **REQ** z priorytetem powiedzmy 6, to proces *i*-ty może wywnioskować:

> Proces *j*-ty ma wyższy priorytet, dostanie ode mnie mój **REQ** i wtedy się dowie, że ja mam priorytet niższy. Będzie więc wiedział, że ja na pewno muszę mu wysłać **ACK**. Będzie więc zachowywał się tak, jakbym już mu **ACK** wysłał, a więc - nie będzie czekać na **ACK** ode mnie. W takim razie, nie muszę mu w ogóle wysyłać **ACK**!

Zaoszczędzamy więc jeden **ACK**! W przypadku szczególnym, gdy wszyscy się ubiegają, odpada więc n **ACK**ów, a proces z najwyższym priorytetem wchodzi w pierwszej rundzie: mamy więc chwilową złożoność czasową 1, a komunikacyjną pakietową tylko *n*!

Zadanie do samodzielnego wykonania

Prowadzący przypisze do każdej dwuosobowej grupy problem dostępu do rozproszonej sekcji krytycznej. Założenia dotyczące środowiska oraz wymogi co do rozwiązania są w pliku “Zasady zaliczenia przedmiotu”.