Algorytm Dekkera

Algorytm Dekkera służy do zrealizowania sekcji krytycznej dla dwóch procesów. Kiedy obydwa procesy ubiegają się o wejście do sekcji krytycznej każdy z nich ustawia zmienną logiczną w pamięci współdzielonej, osobną dla każdego procesu. O tym, który proces wejdzie do sekcji krytycznej jako pierwszy decyduje wartość trzeciej zmiennej współdzielonej przybierającej wartość będącą identyfikatorem jednego z procesów. Drugi proces – ten, którego identyfikator nie jest wartością tej zmiennej współdzielonej - zostaje zawieszony w aktywnej pętli while, dopóki pierwszy proces nie zakończy pracy w sekcji krytycznej i nie zmieni identyfikatora w tej zmiennej współdzielonej. W ten sposób procesy nawzajem wpuszczają się do sekcji krytycznej.

Algorytm Dijkstry

Algorytm Dijkstry służy do zrealizowania sekcji krytycznej dla dowolnej liczby procesów. Kiedy kilka procesów ubiega się o wejście do sekcji krytycznej, każdy z nich ustawią flagę w tablicy indeksowanej numerami tych procesów w pamięci współdzielonej. Następnie procesy ubiegają się o ustawienie zmiennej współdzielonej określającej, który z procesów ma pierwszeństwo wejścia do sekcji krytycznej. Zmienna ta zostanie ustawiona na numer jednego z procesów o ile tylko proces, którego numer jest przechowywany aktualnie w tej zmiennej (ten, który ma aktualnie pierwszeństwo wejścia do sekcji krytycznej). Ten proces wychodzi z aktywnej pętli while i przechodzi do etapu drugiego rozpoczynającego się od zmiany ustawienia flagi we współdzielonej tablicy na numer etapu 2. Jeśli do etapu drugiego przejdzie tylko jeden proces, zostaje on wpuszczony do sekcji krytycznej. Po zakończeniu sekcji krytycznej każdy proces zeruje ustawienia flagi w tablicy współdzielonej.

Algorytm Petersona

Algorytm Petersona dla dwóch procesów służy do zrealizowania sekcji krytycznej dla dwóch procesów. Procesy ubiegające się o wejście do sekcji krytycznej ustawiają zmienne logiczne współdzielone, każdy swoją, oznaczające chęć ubiegania się o wejście do sekcji krytycznej i trzecią zmienna przechowującą numer procesu, który ma pierwszeństwo wejścia do sekcji krytycznej na wartość forującą rywala. Każdy proces następnie w aktywnej pętli oczekuje na swoją kolej w dostępie do sekcji krytycznej – kiedy wartość zmiennej faworyzującej proces będzie równa numerowi tego procesu lub drugi proces wyjdzie z sekcji krytycznej. Po zakończeniu działania w sekcji krytycznej procesu resetuje swoją współdzieloną zmienną logiczną.

Algorytm Lamporta

Algorytm Lamporta służy do zrealizowania sekcji krytycznej dla dowolnej ilości procesów. Procesu ubiegający się o wejście krytycznej najpierw oblicza swój numer w kolejce do sekcji krytycznej. Te numery są przechowywane we współdzielonej tablicy. Do obliczenia tego numeru wykorzystywana jest druga współdzielona tablica, także indeksowana numerami procesów, w której wpis oznacza, czy dany proces jest w trakcie obliczania swojego numeru w kolejce do sekcji krytycznej. Po zakończeniu obliczenia swojego numeru w kolejce proces w dwóch aktywnych pętlach oczekuje na to aż wszystkie procesy zakończą obliczanie swoich numerów w kolejce i wszystkie procesy, które mają wcześniej uzyskać dostęp do sekcji krytycznej. W przypadku identycznego numeru w kolejce dla dwóch procesów, ten, który ma niższy numer uzyskuje wcześniejszy dostęp do sekcji krytycznej. Po zakończeniu działania w sekcji krytycznej proces zeruje swój numer w kolejce.

Algorytm Ricarta i Agrawali

Algorytm Ricarta i Agrawali jest zoptymalizowana wersją algorytmu Lamporta. Każdy proces ma swój zegar logiczny. Jeżeli proces chce wejść do sekcji krytycznej, wysyła do wszystkich innych komunikat

żądania ze swoim numerem, oraz zapamiętuje czas swojego żądania. Do każdego komunikatu dołączony jest stempel czasowy. W przypadku gdy proces otrzyma żądanie to: jeżeli nie chce wejść do sekcji krytycznej, wysyła zgodę żądającemu lub jeśli chciał wejść to porównuje stempel czasowy żądania z czasem swojego żądania, jeśli jego jest wyższy(chciał później uzyskać dostęp) to wysyła żądającemu zgodę. Jeśli nie, dodaje żądającego do kolejki. Jeżeli stempel żądającego jest taki sam jak czas żądania odbiorcy, to odbiorca wysyła zgodę jeśli proces żądający ma mniejszy numer od jego numeru (bez tego byłyby błędy - zakleszczenie lub naruszenie bezpieczeństwa). Gdy proces otrzyma zgody od wszystkich procesów, wchodzi do sekcji krytycznej. Podczas wychodzenia z sekcji krytycznej, proces wysyła zgodę wszystkim procesom, które umieścił wcześniej w kolejce oczekujących. W sekcji wyjściowej roześle zgodę do wszystkich zapamiętanych na liście oczekujących.

Algorytm Maekawy

W pierwszym kroku algorytmu Maekawy proces Pi, który ubiega się o wejście do sekcji krytycznej, rozsyła żądanie do wszystkich procesów, od których wymaga pozwolenia. Kiedy proces Pj otrzyma komunikat żądania, wysyła odpowiedź do procesu Pi pod warunkiem, że nie wysłał już komunikatu z odpowiedzią od czasu otrzymania ostatniej wiadomości zwolnij. W przeciwnym wypadku komunikat z żądaniem trafia do kolejki, w celu jego późniejszego rozpatrzenia. W momencie kiedy proces Pi otrzyma komunikaty typu odpowiedź od wszystkich procesów ze zbioru Ri, może uruchomić swoją sekcję krytyczną. Po wykonaniu sekcji krytycznej, proces Pi wysyła wiadomości zwolnij do wszystkich procesów w Ri. W wypadku kiedy proces Pj otrzyma komunikat zwolnij od procesu Pi, wysyła odpowiedź do następnego oczekującego procesu, który jest w jego kolejce i usuwa go z niej. Jeżeli kolejka jest pusta, wtedy proces aktualizuje swój stan.

Algorytm Raymonda

Algorytm Raymonda używa struktury drzewa którego korzeniem jest proces. Kiedy proces Pi chce wejść do sekcji krytycznej, wysyła żądania wzdłuż ścieżki do korzenia i dodaje żądania do kolejki. Następuje odebranie żądania, wstawienie go do kolejki i przesłanie dalej wzdłuż ścieżki do korzenia. Następnie zostaje wysłany żeton przez korzeń do ubiegającego się procesu oraz aktualizacja zmiennej posiadacz. Żeton zostaje odebrany, następuje wyjęcie żądania z kolejki i wysłanie do proces wskazanego w tym żądaniu, a zmienna posiadacz zostaje zaktualizowana. Proces wchodzi do sekcji krytycznej pod warunkiem, że otrzymał żeton, a jego żądanie jest na szczycie jego kolejki żądań. Zwalnianie sekcji krytycznej następuje poprzez przesłanie żetonu i aktualizacji kolejki oraz zmiennej posiadacz, ewentualnie przesłanie żądania jeżeli kolejka żądań jest niepusta.

Algorytm Tyrana

Algorytm tyrana korzysta z miana elekcji, czyli określenia postępowania prowadzącego do wyboru procesu w celu przejęcia funkcji procesu głównego, który uległ awarii.

Algorytm posiada swoją nazwę z powodu swego działania. Gdy zostanie wznowione działanie, wcześniej uszkodzonego procesu rozpoczyna się elekcja i gdy uzna że jest koordynatorem - procesem o najwyższym id rozsyła tą informację do wszystkich procesów.

Algorytm	**Ilość procesów**	**Wymagane operacje atomowe**	**Ilość pamięci współdzielonej**	**Ilość pamięci lokalnej**	**Możliwość zakleszczenia (deadlock)**	**Możliwość zagłodzenia (livelock)**	**Aktywne pętle**	**Odporność na awarię**	**ilość przesłanych komunikatów**
Dekkera	2	zapis/odczyt do pamięci współdzielonej	Ω(1)	Ω(1)	brak	brak	tak	nie	2
Dijkstry	n	zapis/odczyt do pamięci współdzielonej	Ω (n)	Ω(1)	brak	jeśli inne procesy będą szybsze i będą ustawiać zmienną *turn*	tak	nie	2n
Morrisa	n	operacje *P, V* na semaforach	Ω (1)	Ω(1)	brak	brak o ile dostęp do semaforów jest sprawiedliwy (np. *fifo*)	nie	nie	
Lamporta	n	zapis/odczyt do pamięci współdzielonej	Ω (n)	Ω(1)	brak	brak, przy skończonej ilości procesów	tak	nie	
Petersona dla dwóch procesów	2	zapis/odczyt do pamięci współdzielonej	Ω (1)	Ω (1)	brak	brak	tak	nie	2
Petersona	n	zapis/odczyt do pamięci współdzielonej	Ω (n)	Ω (1)	brak	brak	tak	nie	n^2
Algorytm Centralnego Serwera	n	zapis/odczyt do pamięci współdzielonej	Ω (n)	Ω (1)	brak	brak	tak	nie	2n-1
Algorytm tyrana, (Garcia Moliny)	n	zapis odczyt do pamięci współdzielonej	Ω (1)	Ω (1)	brak	brak	nie	tak	n^2
Ricarta Agrawali	n	zapis/odczyt do pamięci współdzielonej	Ω (1)	Ω (1)	brak	brak	nie	tak	n
Changa Robertsa	n	zapis/odczyt do pamięci współdzielonej	Ω (1)	Ω (1)	brak	brak	nie	tak	3n-1
Szymańskiego	n	zapis/odczyt do pamięci współdzielonej	Ω (n)	Ω (1)	brak	brak	tak	nie	2n-1

1. Opis sposobu oceniania algorytmów i wyboru najlepszego

Ocena algorytmów bez implementacji i możliwości testowania z pewnością nie jest najbardziej optymalnym sposobem na porównanie ich możliwości. Jednak do oceny i wyboru najlepszego algorytmu wzajemnego wykluczania z pewnością należy najpierw sprawdzić czy algorytm na pewno zawsze wykona się poprawni. W naszym zestawienie nie ma algorytmu który mógłby doprowadzić do zakleszczenia ani zagłodzenia, co jest bardzo dobrą wiadomością.
Kolejne kluczowe kryterium to ilością procesów dla których algorytm działa. Algorytmy działające jedynie dla dwóch procesów, nie są wstanie spełnić dzisiejszych oczekiwań, dlatego na wstępie są odrzucane przy dalszym porównaniu.

Kolejną wymaganą cechą jest odporność na awarię. Jest to cecha która wprowadza duża selekcję algorytmów, ponieważ spora część nie spełnia powyższego warunku.

Następne cechy są już mniej krytyczne i związane są z optymalizacją działania algorytmów. Aktywne pętle to są pętle oczekujące na jakiś warunek (najczęściej typu while() np w algorytmie Dijekstry). Kolejnie ilość pamięci współdzielone i lokalnej, czyli jeśli algorytm wykorzystuje współdzieloną/lokalną tablica to jest $\Omega(n)$, a jeśli tylko pojedyncze zmienne to $\Omega(1)$. I ostatecznie ilość przesyłanych komunikatów która w powyżej przedstawionych algorytmach jest bardzo zróżnicowana.

Na podstawie powyższych kryteriów, można wywnioskować, że algorytmy Dekkera i Petersona dla dwóch procesów nie spełniają pierwszych założeń o liczbie n-procesów i zostają odrzucone w dalszej analizie. Kolejnie, algorytmy które mogą ulec zagłodzeniu czyli Dijkstry, Lampota i Morrisa również nie będą dalej rozpatrywane.

Ilość wysyłanych komunikatów przez algorytmy jest bardzo zróżnicowana dlatego teraz rozpatrzymy ten parametr i z pewnością będziemy mogli usunąć algorytm centralnego serwera który potrzebuje za każdym razem aż 2n-1, algorytm tyrana w pesymistycznej wersji jeszcze więcej bo n^2.
Algorytmy Ricarda i Agwarali oraz Szyamańskiego wydają się być jedynymi z najlepszych algorytmów w zestawieniu ale to algorytm Ricarda i Agarali jest tym który jest wykorzystywany w implementacjach częściej, prawdopodobnie ze względu na wysoką wydajność i niezawodność. Dlatego ten algorytm jest naszym zdaniem najlepszy w całym zestawieniu.

Bibliografia:

Wykłady prof. K. Banaś, AGH + źródło elektroniczne
(http://www.metal.agh.edu.pl/~banas/SRR/SRR_W12_Stan_globalny_Koordynacja.pdf)
http://smurf.mimuw.edu.pl/node/948 dostęp 2.05.2016
http://www.algorytm.org/wzajemne-wykluczanie/ dostęp 2.05.2016
Systemy rozproszone podstawy i projektowanie - G. Coulouris, J. Dollimore, T. Kindberg rozdział