Indeks: 272816

Kurs: Systemy Operacyjne 2 [P]

Prowadzący: Inż. Damian Raczkowski

Problem Jedzących Filozofów

Spis treści:

1.Wstęp

Problem jedzących filozofów to klasyczny problem optymalizacyjny, który polega na odpowiedniej synchronizacji wątków i zabezpieczeniu sekcji krytycznych. Celem projektu jest stworzenie programu, ***który unika występowania wyścigów wątków (race condition)*** *, czyli wykorzystania sekcji krytycznych przez różne wątki oraz jest* ***odpory na zakleszczenie (deadlock)*** *czyli niepożądane zablokowanie zasobów. Program symuluje problem jedzących filozofów.*

2.Sposób kompilacji

Plik uruchom_skrypt.sh należy umieścić w tym samym katalogu co plik .cpp, należy nadać mu prawo do wykonywania za pomocą komendy chmod +x uruchom_skrypt.sh, następnie należy go uruchomić za pomocą komendy bash uruchom_skrypt.sh.

Zostanie także utworzony plik wykonywalny o takiej samej nazwie co plik .cpp i należy go uruchomić poleceniem ./Problem5Filozofów

3.Działanie programu

`void philosopher(int id)` – funkcja jako argument przyjmuje id filozofa, symuluje działania filozofa, wywołuje funkcje `aquire_forks()` oraz `put_forks()`. Każdy filozof reprezentuje wątki.

`std::vector<std::mutex> forks(NUM_PHILOSOPHERS);` - `vector forks` reprezentuje widelce, które są mutexami, ponieważ stanowią główną sekcję krytyczną programu.

`std::counting_semaphore<NUM_PERMITS> eating_permit(NUM_PERMITS);` - to semafor, który ogranicza liczbę jedzących filozofów.w

`bool acquire_forks(int id)` – funkcja ma na celu wykonanie próby uzyskania widelców, zwraca wartości `true` lub `false`. Jeżeli filozof uzyska 2 widelce funkcja zwraca `true`, jeżeli nie to `false`. Zwracane komunikaty zależą od wyniku uzyskania widelców. W przypadku gdy filozof uzyska obydwa widelce to wtedy je przez 5 sekund, jeżeli nie może uzyskać żadnego to zaczyna ponownie myśleć.

`void put_forks(int id)` – funkcja zostaje wykonana tylko i wyłącznie w przypadku gdy funkcja `acquire_forks()` zwróciła wartość `true`. Symuluje odłożenie widelców przez filozofa, po czym filozof zaczyna myśleć przez losowy odstęp czasu.

4.Mechanizmy synchronizacji

Każdy **filozof reprezentuje wątki**. W celu zabezpieczenia sekcji krytycznych oraz niedopuszczeniu do zakleszczenia zostały zastosowane następujące mechanizmy:

- Losowy czas wybudzania wątków w celu **symulacji działania rzeczywistego systemu oraz zabezpieczeniu przed możliwym zakleszczeniem** wynikającym z braku dostępnych widelców.

- Zastosowanie semafora aby ograniczyć liczbę jedzących filozofów o połowę w połączeniu z losowym czasem wybudzania, ograniczenie **redukuje ryzyko zakleszczenia**, ponieważ zapewnia, że nie wszyscy filozofowie będą próbować jeść jednocześnie.

- Zastosowanie dodatkowej tablicy `forks_locked()`, która zapisuje czy dany widelec jest wykorzystywany, w przypadku gdy filozof nie może uzyskać 2 widelców zaczyna ponownie myśleć. **Zapobiega to przetrzymywaniu widelca** przez filozofa przez dłuższy okres czasu.

- Zabezpieczenie sekcji krytycznych `vetora forks` oraz tablicy `forks_locked()` przed jedoczesnym dostępem za pomocą mutexów.

- Zabezpieczenie mutexem drukowania na konsolę komend `std::cout` w celu synchronizacji wyjść.

4.1.Dodatkowe informacje:

W danym momencie **filozof może myśleć, jeść albo zakończyć jedzenie** dla każdego z tych stanów jest wyświetlany stosowny komunikat. Komunikaty są wyświetlane w **przypadkach inicjalizacji wątków oraz zmiany stanów filozofa**.

Filozof je przez określony czas **5 sekund**, a myśli w losowym przedziale czasu od **1 sekundy do 15 sekund**.

Symulacja trwa przez **60 sekund**, ale filozofowie **mogą jeszcze zakończyć swoje aktualne czynności** przed zamknięciem programu (np. kończą jeść, jeśli już zaczęli).

5.Podsumowanie i wnioski

Program spełnia założenia projektowe. Przeprowadzone testy wykazały jego poprawne działanie **Wątki są odpowiednio synchronizowane, dostęp do sekcji krytycznych jest zabezpieczony oraz nie występuje problem deadlock.** Program został także uruchamiany dla większej liczby filozofów, jednakże nie polecam ustawiać liczby filozofów na 1 000 000, ponieważ doprowadziło to do kompletnego zawieszenia systemu. Dzięki wykorzystaniu mechanizmu synchronizacji wyjść komunikaty są czytelne.

Lista `forks_locked()` została dodana ze względu na niepoprawne działania metody `try_lock()`, która powodowała błędy. Działanie listy jest przemyślane w ten sposób, że gdy filozof uzyskuje dostęp do widelców najpierw oznacza je w liście jako zablokowane, a potem je bierze. W przypadku odkładania widelców najpierw je odkłada, a potem zostają oznaczone jako dostępne na liście.

Ze względu na to, że w programie została zastosowana losowość, problem zagłodzenie wątków nie powinien być niepokojący.

W kodzie można także zauważyć patent, który był wykorzystywany aby filozof najpierw zabierał widelec z wyższym id, ale po dodaniu listy widelców ten sposób nie ma większego wpływu na działanie programu.

6.Źródła

https://4programmers.net/Forum/C_i_C++/323201-program_problem_pieciu_filozofow
https://pl.wikipedia.org/wiki/Problem_ucztuj%C4%85cych_filozof%C3%B3w
https://pl.wikipedia.org/wiki/Problem_wzajemnego_wykluczania
https://pl.wikipedia.org/wiki/Semafor_(informatyka)
https://cpp0x.pl/dokumentacja/standard-C++11/mutex/1419
https://www.modernescpp.com/index.php/semaphores-in-c-20/
https://www.geeksforgeeks.org/dining-philosophers-problem/
https://www.geeksforgeeks.org/thread-functions-in-c-c/