Sprawozdanie z obowiązkowego zadania: rozwiązanie problemu czytelników i pisarzy

Sławomir Górawski

2 stycznia 2018

1. Treść zadania

Napisz w języku C z użyciem semaforów program synchronizujący procesy. Objaśnij cel (założenia) danej synchronizacji oraz udokumentuj zarówno program, jak i sposób jego testowania.

2. Opis problemu

Problem czytelników i pisarzy to klasyczny problem synchronizacji dostępu do jednego zasobu dwóch rodzajów procesów: dokonujących i niedokonujących w nim zmian. Zasób, zwykle określany mianem czytelni, o dostęp do której ubiegają się procesy z obu grup, jest dzielony między:

- czytelników - procesy niedokonujące zmian w zasobie, których dowolna liczba może korzystać z niego jednocześnie,
- pisarzy - procesy dokonujące zmian w zasobie, które wymagają do niego wyłącznego dostępu.

Rozwiązanie problemu polega na takiej synchronizacji procesów z obu grup, by zapewnić zarówno dzielony dostęp do zasobu przez czytelników, jak i możliwość zajęcia zasobu na wyłączność przez jednego z pisarzy, mogącego dokonać w nim zmian.

Problem czytelników i pisarzy posiada kilka wariantów rozwiązania. Najczęściej spotykane jest rozwiązanie, w którym uprzywilejowani są czytelnicy, zaś pisarze muszą czekać na zwolnienie zasobu przez wszystkie inne procesy. Nieco ciekawszy wariant opisanego problemu zakłada, że liczba miejsc w czytelni jest ograniczona przez znaną stałą M, i ten właśnie został rozwiązany przeze mnie.

Różne implementacje rozwiązań problemu czytelników i pisarzy są stosowane w praktyce m.in. w celu synchronizacji dostępu do baz danych, gdzie operacje wykonywane przez pisarzy można potraktować jako transakcje.

3. Rozwiązanie

Do synchronizacji procesów czytelników i pisarzy użyte zostały dwa semafory: binarny `mutex`, zapewniający wyłączność dostępu do zasobu przez pisarza, oraz zliczający `seats`, ograniczający liczbę procesów, jakie mogą w danym momencie przebywać w czytelni (jego maksymalna wartość ograniczona jest przez M).

Proces czytelnika czeka na wolne miejsce w czytelni, po czym zajmuje je, dokonuje odczytu zasobu i zwalnia zajmowane przez siebie miejsce. Używany jest w nim tylko semafor zliczający `seats`, a implementacja procesu przedstawia się następująco:

```
wait(seats)
ODCZYT ZASOBU
post(seats)
```

Implementacja procesu pisarza jest nieco bardziej złożona. Pisarz czeka na wyłączny dostęp do czytelni, po czym po jego otrzymaniu blokuje kolejne miejsca pozostawiane przez wychodzących czytelników. Dopiero po zajęciu wszystkich miejsc może on dokonać modyfikacji zasobu. Po jej zakończeniu zwalnia on wszystkie miejsca w czytelni, a następnie sygnalizuje zakończenie pracy, pozwalając na otrzymanie dostępu innym procesom. Idea implementacji procesu pisarza w pseudokodzie wygląda tak:

```
wait(mutex)
for i = 1..M
    wait(seats)
MODYFIKACJA ZASOBU
for i = 1..M
    post(seats)
post(mutex)
```

Należy zauważyć, że procesy czytelników nie mają bezpośredniego wpływu na działanie procesów pisarzy – to do tych drugich należy obowiązek zapewnienia sobie warunków do modyfikacji zasobów. Jest to w takim razie rozwiązanie faworyzujące pisarzy. Jego wady to:

- potencjalnie możliwe zagłodzenie czytelników,
- niewykorzystanie w pełni możliwości jednoczesnego dostępu do zasobu przez wielu czytelników – proces pisarza, próbując zapewnić sobie warunki do pracy, blokuje kolejne miejsca, które teoretycznie mogłyby jeszcze zostać zajęte przed opuszczeniem czytelni przez wszystkich czytelników.

Tym niemniej powyższe rozwiązanie powinno zapewnić możliwość dostępu do zasobu przez procesy czytelników i pisarzy zgodnie z założeniami, bez możliwości powstawania zakleszczeń.

4. Testy

Program z rozwiązaniem problemu napisanym w języku C znajduje się w pliku `program.c`. Podczas jego wielokrotnego uruchamiania dla różnych danych nie zaobserwowano występowania zakleszczeń, co pozwala wnioskować, że rozwiązanie jest poprawne.

Ponadto przykładowe komunikaty wypisywane przez program w trakcie odczytu lub modyfikacji zasobu przez konkretny proces zostały zapisane do pliku `output.txt`, po czym przeliczone zostały wystąpienia w nich identyfikatorów konkretnych procesów przy użyciu komendy:

```
cat output.txt | grep ID_PROCESU | wc -l
```

Po przeanalizowaniu komunikatów z pliku o długości 264804 linijek, zawierającego zapis pracy programu dla 4 czytelników, 2 pisarzy i 3 wolnych miejsc, aktywność procesów przedstawia się następująco:

READER 1	58140
READER 2	54652
READER 3	57924
READER 4	59349
WRITER 1	18620
WRITER 2	15961

Tabela 1: Aktywność procesów

Zauważyć można w miarę równomierne rozłożenie dostępu do zasobu zarówno wśród procesów czytelników, jak i pisarzy, co również przemawia za poprawnością rozwiązania.