SOI Koncepcja – Laboratorium 3

Kamil Marszałek 331401

Problem

Synchronizacja dostępu do magazynu o zadanej pojemności przez producentów i konsumentów. Każdy producent oraz konsument jednorazowo zgłaszają gotowość do załadowania lub odebrania towaru z magazynu, ilość tego towaru jest liczbą losową z podanego zakresu. Magazyn może być modyfikowany w jednym momencie tylko przez jednego producenta albo jednego konsumenta. Producent może załadować magazyn tylko, jeśli cała partia towaru się zmieści. Konsument może pobrać towar z magazynu, tylko wtedy, gdy w magazynie znajduje się co najmniej tyle towaru, ile wynosi jego zapotrzebowanie. Magazyn jest reprezentowany przez plik tekstowy. Producenci i konsumenci to natomiast wątki.

Idea

Użyjemy dziedziczenia sekcji krytycznej, dzięki temu uzyskamy lepszą płynność wymiany towaru w magazynie niż jakbyśmy użyli pojedynczego mutexa. Aby rozwiązać ten problem użyjemy trzech semaforów binarnych: mutex – do ochrony sekcji krytycznej (odczyt i zapis do pliku), producer_ready (podniesiony umożliwia producentowi wejście do sekcji krytycznej, opuszczony służy do zatrzymania producentów, jeśli nie warto dokładać teraz do magazynu) oraz consumer_ready (podniesiony umożliwia konsumentowi wejście do sekcji krytycznej, opuszczony służy do zatrzymania konsumentów, jeśli nie warto zabierać teraz towaru z magazynu). Jeśli magazyn jest zajęty do 50% swojej pojemności to wtedy producenci są wpuszczani do sekcji krytycznej, w przeciwnym przypadku konsumenci.

Pseudokod

semafor mutex := 1 // do ochrony sekcji krytycznej
semafor producer_ready := 1 // do zarządzania producentami
semafor consumer_ready := 0 // do zarządzania konsumentami
file store := ‘store.txt’
integer capacity := k // pojemność magazynu zostaje podana jako argument zarówno dla konsumentów, jak i dla producentów

```
begin
producer(a, b, timeout)
        boolean status := false // zmienna informująca czy partia towaru została
wstawiona
        integer amount := randomNumber(a, b) // ilość produkowanego towaru
        repeat
                if (status = true) then
                        amount := randomNumber(a, b) //nowy towar jeśli ostatnio sukces
                        status := false
                end if
                down(producer_ready) // blokada innych producentów
                down(mutex) // wejście do sekcji krytycznej
                integer taken := read(store) // stan magazynu – ilość zajętego miejsca
                if (capacity – taken >= amount) then
                        write(store, taken + amount)
                        status := true
                else status := false
                end if
                up(mutex) // wyjście z sekcji krytycznej
                if (status = true) then
                        if (taken  + amount > capacity  /  2) then
                                up (consumer_ready)  // dalej konsument
                        else
                                up(producer_ready) // dalej producent
                        end if
                else
                        up (producer_ready) // jeśli plik nie został zapisany to oznacza, że
                dalej większą szansę zapisu ma producent niż konsument
                end if
                // Zapisanie statusu operacji wstawienia do magazynu do odpowiedniego
                // pliku
                sleep(timeout) // timeout pomiędzy kolejnymi produkcjami
        until true
end
```

```
begin
consument (c, d, timeout)
    boolean status := false // zmienna informująca czy konsument wyjął ostatni towar
    integer amount := randomNumber(c, d) // zapotrzebowanie
    repeat
        if (status = true)
            amount := randomNumber(c, d) // nowe zapotrzebowanie
            status := false
        end if
        down(consumer_ready) // blokada innych konsumentów
        down(mutex) // wejście do sekcji krytycznej
        integer taken := read(store) //stan magazynu
        if (taken >= amount) then
            write(store, taken - amount)
            status := true
        else status := false
        end if
        up(mutex)
        if (status = true) then
            if (taken  - amount  > capacity  /  2) then
                up (consumer_ready) // teraz konsument
            else
                up(producer_ready) // teraz producent
            end if
        else
            up(consumer_ready) // jeśli konsumentowi nie udało się
        zmodyfikować magazynu to niech odda sekcję kolejnemu konsumentowi
        end if
        // zapis statusu operacji do odpowiedniego loggera
        sleep(timeout) // przerwa między kolejnymi generacjami
    until true
end
```

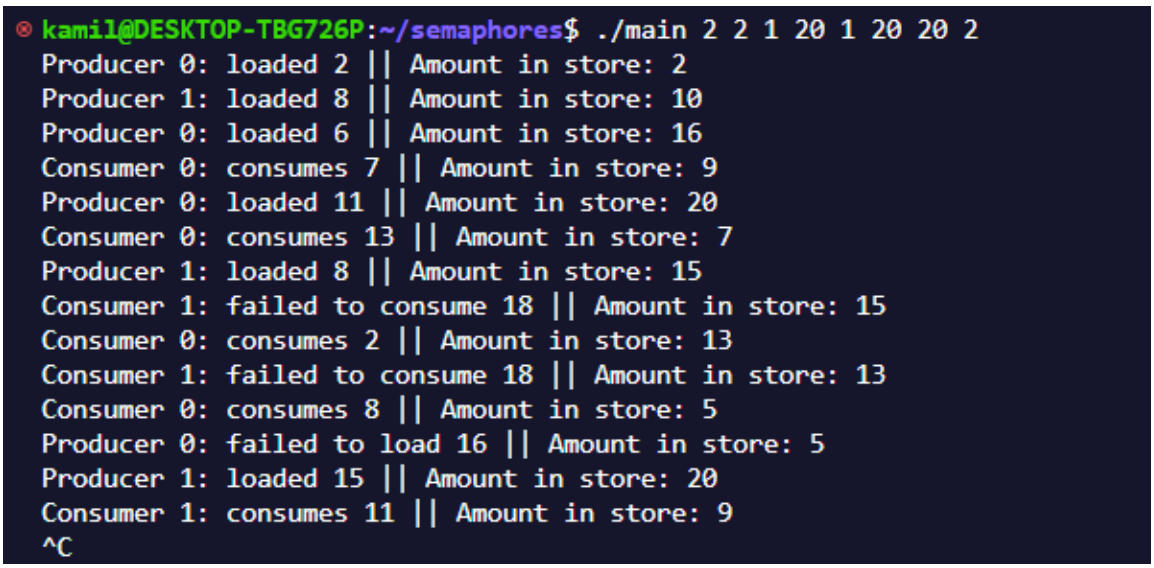
Adnotacja: w przedstawionym kodzie istnieje ryzyko, że jeśli producentów lub konsumentów będzie mało, to może dojść do sytuacji, że jeżeli będzie np. jeden producent, który chce wstawić 14 sztuk towaru, a pozostało tylko miejsca w magazynie na 11 sztuk, to wtedy program będzie krążył w kółko. Aby rozwiązać ten problem

dodałem zliczanie, ile razy producent lub konsument podejmują próbę modyfikacji magazynu. Jeśli liczba nieudanych prób będzie zbyt duża, to partia towaru lub zapotrzebowania na towar zostanie porzucona, zostanie wygenerowana nowa ilość.

Przykładowe wykonania:

Producentów niech będzie dwóch: P0 i P1. Konsumentów również niech będzie dwóch: K0 i K1. Zarówno producenci, jak i konsumenci generują towar lub zapotrzebowanie na towar losując liczbę z zakresu od 1 do 20. Pojemność magazynu wynosi 20. Na samym początku wykonania magazyn jest pusty. Zatem, któryś z producentów dostanie szansę załadować do magazynu. P0 pierwszy zamyka semafor producer_ready, a potem zamyka mutex i wchodzi do sekcji krytycznej. Chce on dołożyć do magazynu 2 sztuki towaru, udaje mu się to. Wychodzi z sekcji krytycznej (podnosi mutex). Pozostałe wątki są zawieszone na dwóch pozostałych semaforach consumer_ready i producer_ready. Następnie na podstawie tego czy została zajęta już połowa magazynu producent P0 oddaje sekcję krytyczną albo producentom, albo konsumentom. W tym przypadku możliwość wypełnienia magazynu dostanie P1. Próbuje załadować magazyn 8 sztukami towaru. Udaje mu się to, nowy stan magazynu to 10. Magazyn jest dokładnie w połowie pełny, więc szansę dostanie ponownie producent P0. Chce on załadować 6 sztuk towaru, udaje mu się to. Nowy stan magazynu to 16. Sekcja krytyczna zostaje przekazana konsumentom. Konsument K0 pierwszy zamyka semafory. Chce on pobrać z magazynu 7 sztuk towaru udaje mu się to. Nowy stan magazynu to 9, zatem czas dostanie producent. Producent P0 pierwszy wchodzi do sekcji krytycznej. Chce on włożyć do magazynu 11 sztuk towaru udaje mu się to. Nowy stan magazynu: 20. Następnie pobudzony zostaje konsument, tym razem jest to K0. Chce on pobrać 13 sztuk towaru udaje mu się to, stan magazynu 7. Mutex zostaje otwarty i możliwość wejścia producent tym razem P1. Na tym poprzestanę załączam jeszcze test:

Producenci: 2, Konsumenci: 2, Magazyn: 20, zakres produkcji/zapotrzebowania (1 – 20)



Przyjrzyjmy się jeszcze co będzie się działo w przypadkach skrajnych:

Jeśli producentów będzie znacznie więcej niż konsumentów, wtedy do sekcji krytycznej pojedynczy wątek konsumenta będzie wchodził znacznie częściej niż pojedynczy wątek producenta. Taki mechanizm jest zapewniony dzięki przekazywaniu sekcji krytycznej tym wątkom, które mają większą szansę na sukces w modyfikacji stanu magazynu. W tej sytuacji, jeśli nie byłoby dziedziczenia sekcji krytycznej to trochę czasu mogłoby upłynąć zanim konsument wszedłby do magazynu, a producenci wychodziliby z magazynu nie odnosząc sukcesu. Byłoby to dużo mniej efektywne. Podobnie można rozważyć sytuację, jeśli to konsumentów jest znacznie więcej niż producentów.

Producenci: 1, Konsumenci: 10, Magazyn: 20, zakres produkcji/zapotrzebowania (1 – 20)

```
kamil@DESKTOP-TBG726P:~/semaphores$ ./main 1 10 1 20 1 20 20 2
Producer 0: loaded 2 || Amount in store: 2
Producer 0: loaded 14 || Amount in store: 16
Consumer 0: consumes 7 || Amount in store: 9
Producer 0: failed to load 15 || Amount in store: 9
Producer 0: failed to load 15 || Amount in store: 9
Producer 0: loaded 6 || Amount in store: 15
Consumer 1: failed to consume 18 || Amount in store: 15
Consumer 2: failed to consume 16 || Amount in store: 15
Consumer 3: consumes 14 || Amount in store: 1
Producer 0: loaded 5 || Amount in store: 6
Producer 0: loaded 1 || Amount in store: 7
^C
```

Widzimy również, że dodanie mechanizmu kontroli ilości prób pomaga w utrzymaniu odpowiedniej płynności działania. Jeśli producent nie jest w stanie w dwóch próbach zmodyfikować magazynu, wtedy zostaje wygenerowana nowa wartość produkcji.

Producenci: 10, Konsumenci: 1, Magazyn: 20, zakres produkcji/zapotrzebowania (1 – 20)

```
kamil@DESKTOP-TBG726P:~/semaphores$ ./main 10 1 1 20 1 20 20 2
Producer 0: loaded 2 || Amount in store: 2
Producer 1: loaded 17 || Amount in store: 19
Consumer 0: consumes 3 || Amount in store: 16
Consumer 0: failed to consume 20 || Amount in store: 16
Consumer 0: failed to consume 20 || Amount in store: 16
Consumer 0: consumes 4 || Amount in store: 12
Consumer 0: consumes 7 || Amount in store: 5
Producer 2: loaded 10 || Amount in store: 15
Consumer 0: consumes 1 || Amount in store: 14
Consumer 0: consumes 13 || Amount in store: 1
Producer 3: loaded 8 || Amount in store: 9
^C
```

Można zauważyć, że gdyby konsument po kilku nieudanych próbach, nie porzucał swojej partii to magazyn nie byłby modyfikowany, program stałby w tym samym miejscu.

Można również zauważyć, że zaletą wykorzystania przekazywania sekcji krytycznej, jest praktycznie stu procentowa skuteczność przepływu towaru przez magazyn. Prawie nie ma nieudanych prób.

Producenci: 3, Konsumenci: 3, Magazyn: 50, zakres produkcji/zapotrzebowania (1 – 20)

```
kamil@DESKTOP-TBG726P:~/semaphores$ ./main 3 3 1 20 1 20 50 2
Producer 0: loaded 2 || Amount in store: 2
Producer 1: loaded 17 || Amount in store: 19
Producer 0: loaded 11 || Amount in store: 30
Consumer 0: consumes 16 || Amount in store: 14
Producer 2: loaded 10 || Amount in store: 24
Producer 1: loaded 13 || Amount in store: 37
Consumer 1: consumes 14 || Amount in store: 23
Producer 0: loaded 16 || Amount in store: 39
Consumer 1: consumes 11 || Amount in store: 28
Consumer 0: consumes 2 || Amount in store: 26
Consumer 2: consumes 16 || Amount in store: 10
Producer 2: loaded 15 || Amount in store: 25
Producer 1: loaded 14 || Amount in store: 39
Consumer 1: consumes 4 || Amount in store: 35
Consumer 0: consumes 7 || Amount in store: 28
Consumer 1: consumes 17 || Amount in store: 11
Producer 0: loaded 8 || Amount in store: 19
Producer 2: loaded 1 || Amount in store: 20
Producer 0: loaded 16 || Amount in store: 36
Consumer 2: consumes 1 || Amount in store: 35
Consumer 0: consumes 12 || Amount in store: 23
^C
```