Synchronizacja procesów z wykorzystaniem monitorów

Autor: Halyna Polekha

Problem

Producent-konsument, przy następujących założeniach:

- występuje jeden "producent", który generuje zadania
- oraz trzech "konsumentów" A, B, C, którzy czytają i usuwają elementy z bufora
- do komunikacji jest wykorzystywany 1 bufor
- element jest usuwany z bufora, jeżeli zostanie przeczytany przez albo obu konsumentów A i B, albo przez obu konsumentów B i C
- konsument A nie może przeczytać elementu, jeżeli został on już przez niego wcześniej przeczytany, albo został przeczytany przez konsumenta C i na odwrót.

Założenia wstępne

- nie dopuścić do czytania z pustego bufora,
- nie dopuścić do zapisu do pełnego bufora,

Opis struktury reprezentujących bufor komunikacyjny

Do obsługi kolejki bufora będę stosować monitor Buffer zawierający implementacje FIFO (std::deque), mutex realizujący wyłączny dostęp do monitora oraz warunki do czasowego zwolnienie wyłącznego dostępu.

```
class Buffer : public Monitor { // dziedziczy klasyczny monitor (z muteksem oraz funkcjami
                                                          wyłącznego dostępu)
 private:
        deque<char> buffer; // implementacja FIFO
        char product; // flaga do opisu „kto przeczytał ” pierwszy element z głowy kolejki

        Condition sem_not_empty; // warunek opisujący czy w kolejce znajduje się chociaż
                                                  jeden element (czy bufor nie jest pusty)
        Condition sem_not_full;  // warunek opisujący czy ilość elementów w kolejce jest
                                          mniejsza od 9 (czy bufor nie jest pełny)
        Condition sem_let_A; // warunek zwolnienia konsumera A ze stanu zawieszenia i
                                                          pozwolenia na czytanie
        Condition sem_let_B; // analogicznie dla Konsumera B
        Condition sem_let_C; // analogicznie dla Konsumera C

        void update_product(); // do aktualizacji flagi
        void push(char prod) ; // do wstawienia elementa na koniec kolejki
        void pop() // do usuwania elementa „z głowy” kolejki
 public:
 void produce(); // funkcja producenta
 void consume_A(); // funkcja konsumenta A
 void consume_B(); // konsumenta B
 void consume_C(); // konsumenta C
```

};

Działanie programu

Producent będzie zgłaszał zadanie, które będzie padało do kolejki, jeśli ta nie jest pełna.

Te zadania będzie odbierał jeden z trzech konsumentów. Przy czym w taki sposób, że będzie pobierał zadanie z kolejki wyłącznie po poprzednim przeczytaniu (modyfikacji flagi product) przez odpowiedniego konsumenta.

Pseudokod przedstawiający synchronizację procesów

Producent

1. Poczekaj na podniesienie semafora monitora , opuść i wejdź do SK (**enter()**)
2. Czy kolejka nie jest pełna? Jeżeli jest to wyjdź z SK, czekaj aż zwolni się miejsce **wait(warunek)** i kontynuuj działanie
3. Wstaw element do kolejki (na koniec)
4. Zasygnalizuj że kolejka nie jest pusta
5. Opuść SK **leave()**
6. Po wysłaniu odpowiedniej liczby sygnałów – zakończ program

Konsument

1. Poczekaj na podniesienie semafora monitora , opuść i wejdź do SK (**enter()**)
2. Czy kolejka nie jest pusta? Jeżeli jest to wyjdź z SK, czekaj aż przestane być pusta **wait(warunek)** i kontynuuj działanie
3. Sprawdź flagę elementu z głowy kolejki i odpowiednio do warunków zadania usuń element, ustaw flagę lub zawieś się do momentu pozwolenia na kontynuację.
4. Jeżeli kolejka nie jest pełna (mniej niż 9 elementów) to zasygnalizuj
5. Opuść SK **leave()**
6. Po obsłużeniu określonej liczby sygnałów – zakończ działanie

Main

1. Producent zaczyna działanie
2. Konsumenci zaczynają działanie
3. Poczekaj na zakończenie wszystkich procesów potomnych
4. zakończ