SOI – Semafory

Koncepcja

1. Zadanie

Mamy bufor FIFO na liczby całkowite. * Procesy A1 generują kolejne liczby parzyste modulo 50, jeżeli w buforze jest mniej niż 10 liczb parzystych. * Procesy A2 generują kolejne liczby nieparzyste modulo 50, jeżeli liczb parzystych w buforze jest więcej niż nieparzystych. * Procesy B1 zjadają liczby parzyste pod warunkiem, że bufor zawiera co najmniej 3 liczby. * Procesy B2 zjadają liczby nieparzyste, pod warunkiem, że bufor zawiera co najmniej 7 liczb. W systemie może być dowolna liczba procesów każdego z typów. Zrealizuj wyżej wymienioną funkcjonalność przy pomocy semaforów. Zakładamy, że bufor FIFO poza standardowym put() i get() ma tylko metodę umożliwiającą sprawdzenie liczby na wyjściu (bez wyjmowania) oraz posiada metody zliczające elementy parzyste i nieparzyste. Zakładamy, że semafory mają tylko operacje P i V.

2. Schemat Ogólny

W zadaniu mamy cztery rodzaje procesów (A1, A2, B1, B2) działających na wspólnym buforze FIFO. Kluczowym jest użycie semaforów do synchronizacji tych procesów, z zachowaniem określonych warunków.

3. Struktury i Semafory

- Bufor FIFO: Struktura do przechowywania liczb całkowitych z metodami put(), get(), peek(), countEven() i countOdd().
- Semafor Licznikowy: Dwa semafory licznikowe – jeden dla liczb parzystych (evenSemaphore), drugi dla nieparzystych (oddSemaphore).
- Mutex: Semafor binarny do zarządzania dostępem do bufora.

4. Logika Procesów

- A1: Sprawdza, czy liczba parzystych jest mniejsza niż 10. Jeśli tak, dodaje kolejną liczbę parzystą modulo 50.
- A2: Sprawdza, czy liczb parzystych jest więcej niż nieparzystych. Jeśli tak, dodaje kolejną liczbę nieparzystą modulo 50.
- B1: "Zjada" liczbę parzystą, jeśli w buforze jest co najmniej 3 liczby.
- B2: "Zjada" liczbę nieparzystą, jeśli w buforze jest co najmniej 7 liczb.

5. Testowanie

- Testy Jednostkowe: Sprawdzenie poprawności każdej operacji (put, get, countEven, countOdd) na buforze.
- Testy Wielowątkowości: Uruchomienie równocześnie wielu instancji każdego rodzaju procesów i sprawdzenie czy bufor zachowuje się zgodnie z wymaganiami.
- Testy Wydajnościowe: Sprawdzenie, jak system radzi sobie pod dużym obciążeniem, czy nie dochodzi do zakleszczeń.

6. Szkic implementacji

```
#include <semaphore>
#include <mutex>
#include <thread>

class FifoBuffer {
   // Implementacja metod bufora
};

std::counting_semaphore<1> bufferAccess(1); // Semafor dla dostępu do bufora
std::counting_semaphore<10> itemsAvailable(0); // Semafor dla dostępnych elementów
std::counting_semaphore<10> evenItemsAvailable(0); // Semafor dla parzystych
std::counting_semaphore<10> oddItemsAvailable(0);  // Semafor dla nieparzystych

FifoBuffer buffer;

void processA1() {
   while (true) {
      bufferAccess.acquire();
      if (buffer.countEven() < 10) {
         buffer.put(generateEvenNumber());
         itemsAvailable.release();
         evenItemsAvailable.release();
      }
      bufferAccess.release();
   }
}

void processA2() {
   while (true) {
      bufferAccess.acquire();
      if (buffer.countEven() > buffer.countOdd()) {
         buffer.put(generateOddNumber());
         itemsAvailable.release();
         oddItemsAvailable.release();
      }
      bufferAccess.release();
   }
}

void processB1() {
   while (true) {
      evenItemsAvailable.acquire();
      bufferAccess.acquire();
      if (buffer.count() >= 3 && isEven(buffer.peek())) {
         buffer.get();
      }
      bufferAccess.release();
   }
}
```

```
void processB2() {
    while (true) {
        oddItemsAvailable.acquire();
        bufferAccess.acquire();
        if (buffer.count() >= 7 && !isEven(buffer.peek())) {
            buffer.get();
        }
        bufferAccess.release();
    }
}
int main() {
    // Kod do uruchomienia procesów
}
```