Treść zadania

Treść zadania jest identyczna jak w przypadku zadania 3, jednak tym razem należy skorzystać z monitorów.

Planowana implementacja

Aby poprawnie zaimplementować zadanie stworzę klasę monitora, a w niej zaimplementuje odpowiednie metody, niezbędne do poprawnej synchronizacji oraz wykonywania operacji na buforze.

Będzie ona posiadać obiekt klasy `Monitor` oraz obiekty klasy `Condition` z pliku `monitor.h`. Są one niezbędne do synchronizowania operacji wykonywanych na buforze.

Będzie posiadała bufor, identyczny jak w moim rozwiązaniu zadania t3.

Zaimplementuję w niej metody sprawdzające podane warunki dla producentów i konsumentów oraz funkcję produkujące i konsumujące liczby parzyste i nieparzyste.

Każda z procedur produkowania i konsumowania będzie zaimplementowana zgodnie z zasadami operacji monitorowych przedstawionych w dokumencie wprowadzającym. Są one również analogiczne do tych używanych w rozwiązaniu zadania t3.

Przykład jednej z planowanych funkcji:

```
void produceOdd() {
    monitor.enter();
    if (!canProduceOdd()) {
        monitor.wait(condProduceOdd);
    }

    // Add to buffer

    if(canProduceEven())
    {
        monitor.signal(condProduceEven);
    }

    // Analogicznie dla wszystkich innych warunków

    monitor.leave();
}
```

Takie rozwiązanie powinno zapewnić bezproblemową synchronizację kilku wątków ubiegających się o dokonywanie operacji na buforze.

Testowanie

Aby przetestować rozwiązanie planuje zrobić to samo, co przy testowaniu zadania t3. To znaczy uruchomić kilka wątków, które będą na zmiane próbowały wykonywać operację na buforze. Scenariusze testowe pozostaną te same, to znaczy

- uruchomienie po jednym producencie i po jednym konsumencie
- uruchomienie po dwóch producentów i konsumentów
- uruchomienie wszystkich konsumentów oraz producentów na raz

Autor

Konrad Wojda, 310990