Wojciech Sitek

Koncepcja SOI – zadanie 4

# Treść zadania

* Napisać program w języku C++, wykorzystujący działanie **monitorów** do symulacji bufora z producentami i konsumentami
* Dany jest **jeden bufor** 9-elementowy FIFO, **dwóch konsumentów** oraz **dwóch producentów**
* Producent A produkuje jedną literę
* Producent B produkuje trzy litery
* Konsument A czyta jedną literę
* Konsument B czyta dwie litery
* Element jest usuwany z bufora po odczytaniu przez konsumenta
* Liczba elementów w buforze po przeczytaniu przez dowolnego konsumenta **nie może spaść poniżej 3**.

# Koncepcja rozwiązania zadania

1. Wykorzystane klasy i narzędzia:

* Wykorzystanie procesów w systemie Linux do implementacji jednoczesnego działania producentów i konsumentów - tworzonych za pomocą funkcji **fork()** i usuwanych za pomocą funkcji **kill().**
* Wykorzystanie biblioteki **Boost** do utworzenia mapy pamięci współdzielonej między procesami, w której znajdować się będą monitor, zmienne warunkowe oraz wspólny bufor.
* Implementacja wirtualnej **klasy** **Person**, która zawiera:
  + Nazwę osoby: Consumer(-)/Producer(+) + A/B, potrzebną do wyświetlenia informacji o niej w programie.
  + Zmienną *jump*: ile liter konsumuje/produkuje osoba naraz
  + Wirtualną metodę *action*, implementowaną w klasach pochodnych Consumer/Producer, które dziedziczą po klasie Person.
* Implementacja **klasy Buffer** dla typu char (bufor 9-elementowy FIFO) dziedziczącej po klasie Monitor oraz następującymi metodami obsługującymi klasę:
  + void add(char) – dodanie elementu na koniec kolejki
  + char pick() – usunięcie pierwszego elementu z kolejki
  + void consume(Consumer \*c):
    - enter();
    - if(too\_low\_num\_elements) => wait(full);
    - *repeat c->jump times:* pick(); inform\_user;
    - if(now\_not\_too\_much\_elements) => leave(); signal(empty);
    - else =>leave();
  + void produce(Producer \*c, char \*letters):
    - enter();
    - if(too\_much\_elements) => wait(empty)
    - *repeat c->jump times:* add(letter); inform\_user();
    - if(now\_not\_too\_low\_num\_elements) => leave(); signal(full);
    - else => leave();
* Pobranie klas Semaphore, Condition oraz Monitor z pliku monitor.h ze strony dra Tomasza Kruka.

1. Opis mechanizmów wzajemnego wykluczania:

* W programie będą używane **dwa monitory**:
  + Pierwszy, będący polem klasy Buffer (zostanie utworzony tylko 1 obiekt klasy Buffer), realizujący wzajemne wykluczanie sekcji krytycznej bufora.
  + Drugi, nazwany *userMutex*, będący realizacją wzajemnego wykluczania komunikacji z użytkownikiem.
* W każdej metodzie bufora na jej początku będzie znajdować się wejście do monitora, a na końcu wyjście z monitora.
* Przed każdym wypisaniem działania programu przez jeden z procesów, będzie się znajdować wejście do monitora *userMutex*, a po wypisaniu wyjście z sekcji monitora.
* Będą zdefiniowane **dwie zmienne warunkowe** (typu Condition):
  + empty – ilość pustych miejsc w buforze
  + full – ilość pełnych miejsc w buforze
* Implementacja metod Consumer.action(), Producer.action():
  + Obie funkcje działają w pętli nieskończonej, zakończonej zakończeniem procesu przez funkcję *kill()* procesu nadrzędnego.
  + Konsument wykonuje w pętli kolejne operacje:
    - buffer.consume(\*this);
  + Producent wykonuje w pętli kolejne operacje:
    - produce\_items;
    - buffer.produce(\*this, items);
* Program wypisuje następujące dane:
  + Producer(+) lub Consumer(-)
  + A lub B
  + [która litera]/[łączny jump]
  + skonsumowana/wyprodukowana litera
  + ilość elementów w buforze
  + cały bufor
* Przykład danych wypisywanych przez programy:
  + + A 1/1 f 4 cdef
  + - B 1/2 c 3 def