Jakub Kowalczyk 318676

Laboratorium 4 - synchronizacja z wykorzystaniem monitorów

Konstrukcja buforów z monitorami

Plik buffer.h zawiera definicję klasy Buffer - monitorowego bufora, który przechowuje składniki potrzebne do produkcji pierogów. Klasa ta dziedziczy po klasie Monitor i posiada składowe:

- nazwa typu składnika jako std::string,
- warunek pełnego bufora (full) i pustego (empty),
- kolejka FIFO wartości logicznych reprezentująca bufor.

Klasa posiada metody produkcji i konsumpcji, które odpowiednio umieszczają i pobierają elementy z bufora. Szczegółowe działanie zostało opisane komentarzami w kodzie. Klasa posiada także gettery potrzebne do printowania informacji.

Producent - Konsument

W porównaniu do implementacji z semaforami, kod funkcji producenta i konsumenta w pliku main.cpp znacznie się uprościł. Podobnie jak poprzednio, obie te funkcje przyjmują jako argument wskaźnik na void, który jest rzutowany na wskaźnik odpowiedniego rodzaju bufora. W nieskończonych pętlach funkcja produce_ingretients() wywołuje jedynie metodę produce() bufora, zaś funkcja consume_for_dumplings() wywołuje metodę consume() na buforze ciasta oraz nadzienia pieroga.

Main

Zgodnie z zaleceniami program przyjmuje 7 argumentów - liczbę wątków dla każdego rodzaju producenta i konsumenta. Po ich wczytaniu zliczam łączną wielkość i tworzę tablice wątków producentów i konsumentów. Następnie w 7 pętlach inicjalizuje wątki za pomocą pthread_create: do tablicy przypisywane jest ich id, NULL jako parametr oznacza brak atrybutów; podaję także funkcję producenta/konsumenta i jej argument - odpowiedniego rodzaju bufor z monitorem. Przed uruchomieniem funkcji pthread_join() indeksy wątków są umieszczane w wektorze i wymieszane losowo za pomocą std::shuffle() z biblioteki algorithm. Zapewnia to startowanie wątków producentów i konsumentów w kolejności losowej.

Testy

W programie zdefiniowane jest zmienna globalna SLOW_MODE, za pomocą której można aktywować funkcję sleep(2) na wątkach, symulując ich powolne działanie.

Zrezygnowałem z opcji printowania zawartości bufora / umieszczania do niego składników przez producentów, gdyż jest ona źle zsynchronizowana z poprawnym działaniem wątków.

Być może, problem rozwiązałoby wprowadzenie mutexu, lecz jest to zadanie z synchronizacją z monitorami, więc zależało mi na wykonaniu go w całości bez użycia semaforów. Na rzecz tego kontroluję stan przepełnienia bufora, z racji że kolejka nie ma stricte określonego rozmiaru. Jeżeli któryś z elementów umieszczonych w kolejce przekroczy umowny rozmiar kolejki BUFFER_SIZE - wyprintuje się informacja o przepełnieniu bufora.

Z usypianiem czy też bez, widać, że konsumpcja i produkcja różnego rodzaju pierogów/składników występuje bez żadnego przerwania i przepełniania buforów. Równomiernie, produkowane są wszystkie rodzaje pierogów.

```
Konsumpcja: pierogi ze składnikiem ser
Konsumpcja: pierogi ze składnikiem mięso
Konsumpcja: pierogi ze składnikiem ser
Konsumpcja: pierogi ze składnikiem mięso
Konsumpcja: pierogi ze składnikiem kapusta
Konsumpcja: pierogi ze składnikiem kapusta
Konsumpcja: pierogi ze składnikiem ser
Konsumpcja: pierogi ze składnikiem mięso
Konsumpcja: pierogi ze składnikiem mięso
Konsumpcja: pierogi ze składnikiem kapusta
Konsumpcja: pierogi ze składnikiem mięso
Konsumpcja: pierogi ze składnikiem ser
Konsumpcja: pierogi ze składnikiem ser
Konsumpcja: pierogi ze składnikiem mięso
Konsumpcja: pierogi ze składnikiem ser
Konsumpcja: pierogi ze składnikiem ser
Konsumpcja: pierogi ze składnikiem mięso
Konsumpcja: pierogi ze składnikiem kapusta
Konsumpcja: pierogi ze składnikiem mięso
Konsumpcja: pierogi ze składnikiem mięso
Konsumpcja: pierogi ze składnikiem ser
Konsumpcja: pierogi ze składnikiem kapusta
```

W związku tym spełnione są podstawowe założenia synchronizacji. Nie występuje jednoczesny zapis do bufora, czytanie z pustego bufora czy też wzajemne “przeszkadzanie sobie” wielu wątków przy zapisywaniu lub czytaniu z bufora.