SOI lab3 koncepcja

Tobiasz Kownacki

Zadanie

Celem zadania jest napisanie programu tworzącego odpowiednie procesy na osobnych wątkach wykonujących różne typy operacji na wspólnym buforze liczbowym.

- 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. Synchronizacja ma być zaimplementowana z wykorzystaniem 5-ciu semaforów binarnych.

Koncepcja

Do synchronizacji będziemy potrzebować pięciu semaforów. Jeden semafor będzie odpowiadać za dostęp do bufora. Będzie on chronić przed sytuacją gdzie jednocześnie są wykonywane różne akcje na pamięci bufora przez kilka wątków na raz. Tylko jeden wątek będzie mógł pracować jednocześnie na buforze. Ten semafor od strefy krytycznej będzie początkowo odblokowany. Pozostałe 4 semafory będą przypisane do każdego konsumenta i producenta. Każdy z tych 4 semaforów będzie początkowo zablokowany. W funkcji odpowiadającej za producentów/konsumentów będzie nieskończona pętla. Jako bufora danych użyje kontenera deque, który jest kolejką z możliwością iterowania po każdym elemencie. Schemat konsumenta/producenta w jednym obiegu nieskończonej pętli:

1. Używamy metody P na semaforze odpowiadającym za dostęp do bufora. Metoda blokuje semafor, a jeśli już wcześniej był zablokowany to wsadza on proces do swojej własnej kolejki procesów i blokuje jego działanie.
2. Producent/Konsument sprawdza czy może wykonać swoją akcje, wynikającą z treści zadania. Jeśli tak przechodzi do pkt. 3. Jeśli nie, zwiększa liczbe czekających produktów/konsumentów, używa metody V na semaforze pamięci bufora, która jest podobna do P, ale odwrotna. Usuwa z kolejki czekający proces, a jeśli takiego nie ma to odblokowuje semafor. Następnie używa metody P na semaforze tego konsumenta/producenta. Potem zmniejszy liczbę czekających konsumentów/producentów tego rodzaju
3. Producent/Konsument wykonuje swoją akcje.
4. Sprawdza dla pozostałych konsumentów/producentów czy ich warunki działania są prawdziwe i któryś z nich czeka w kolejce. Jeśli tak to używa metody V na semaforze danego konsumenta/producenta i przejdź do pkt 1. Jeśli nie to sprawdź kolejnego konsumenta/producenta. Jeśli nie udało się takiego znaleźć to użyj metody V na semaforze od strefy krytycznej.

Testy

Aby przetestować rozwiązanie każdy producent/konsument będzie wypisywał na konsole swoje akcje oraz zawartość bufora. Przydatny będzie sleep(random) po każdym obiegu pętli nieskończonej aby łatwiej przeanalizować co się dzieje w aplikacji. Konieczne będzie również użycie dodatkowej flagi przy kompilacji rozwiązania w c++ -lpthread. Testy będą rozpatrywane dla poniższych przypadków:

1. 1-A1
2. 1-A2
3. 1-B1
4. 1-B2
5. 1-A1, 1-A2
6. 1-B1, 1-B2
7. 1-A1, 1-B1
8. 1-A2, 1-B2
9. 1-A1, 1-A2, 1-B1, 1-B2
10. 2-A1, 2-A2, 2-B1, 2-B2

Użytkownik jako argument programu będzie podawał numer przypadku który chce przetestować.