Monitory

Jakub Robaczewski

Implementacja:

Wiadomość:

W moim rozwiązaniu zaimplementowałem wiadomość jako klasę posiadającą pola:

std::deque Messages - dwustronna kolejka do przechowywania wiadomości

float createtime – czas stworzenia wiadomości

float semaphoretime – czas dodania wiadomości do kolejki

float readtime – czas odczytania wiadomości

Kolejka:

Kolejka została zaimplementowana jako klasa dziedzicząca z klasy Monitor, by implementowała funkcja potrzebne do działania. Sama klasa Monitor wykorzystuje semafory z biblioteki <semaphore.h>.

enter() - wejdź do sekcji krytycznej

leave() - wyjdź z sekcji krytycznej

signal() - obudź procesy czekające na zmianę warunku

wait() - poczekaj na warunek

Na użytek kolejki stworzone zostały 2 warunki (klasa Condition)

- NotFull – w kolejce jest jeszcze miejsce
- NotEmpty – w kolejce są jakiekolwiek wiadomości

Funkcja testująca make_test():

Funkcja testująca make_test() jest głównym sposobem testowania monitorów. By ją zastosować należy stworzyć scenariusz testu: podać parametry procesów (liczebność, liczbę generowanych wiadomości), długość kolejki oraz tabele z czasami uśpienia dla poszczególnych procesów.

Testowanie:

Program jest testowany za pomocą programu main. Program wykonuje 5 testów (sytuacji). Podane są też czasy opóźnienia procesów (Writer, WriterVIP, Reader) oraz długość kolejki.

```
Test 1: Reading from empty queue. [2,0,0] queue 5
18016n0 (2.00057, 2.00061, 2.00064)
18016n1 (2.00069, 2.0007, 2.00075)
18016n2 (2.0007, 2.0007, 2.00123)
18016n3 (2.0007, 2.0007, 2.00129)
18016n4 (2.0007, 2.0007, 2.00132)
18016n5 (2.0007, 2.00071, 2.00143)
18016n6 (2.00071, 2.00079, 2.00144)
18016n7 (2.0008, 2.00125, 2.00145)
18016n8 (2.00126, 2.0013, 2.00145)
18016n9 (2.0013, 2.00133, 2.00147)
```

Test pierwszy sprawdza, czy monitor prawidłowo blokuje proces Reader, przed odczytaniem z pustej tablicy. Jak widzimy mimo że proces Reader uruchamia się natychmiast to wiadomości są odczytywane dopiero po upłynięciu 2 sekund i uruchomieniu procesów pisarzy.

```
Test 2: Reading from full queue. [0,0,2] queue 5
15312n0 (0.000143596, 0.000152188, 2.00015)
15312n1 (0.000153232, 0.000154764, 2.00019)
15312n2 (0.000155208, 0.000156548, 2.0002)
15312n3 (0.000156904, 0.000158184, 2.00021)
15312n4 (0.0001585, 0.000159716, 2.00022)
15312n5 (0.000160032, 2.00016, 2.00028)
15312n6 (2.00016, 2.0002, 2.00029)
15312n7 (2.0002, 2.00021, 2.00029)
15312n8 (2.00021, 2.00022, 2.00029)
15312n9 (2.00022, 2.00023, 2.00029)
```

Test drugi sprawdza, czy monitor prawidłowo blokuje procesy pisarzy przed dodawaniem wiadomości do pełnej kolejki. Pierwsze 5 procesów dodaje wiadomości do kolejki, a następny tworzy ją i oczekuje na odblokowanie kolejki, które następuje dopiero 2 sekundy później, po uruchomieniu procesu Reader.

```
Test 3: Only normal writers, last vip. [0,1,3] queue 6
15312v0 (1.00015, 1.00016, 3.00047)
15312v1 (1.00016, 3.00051, 3.00059)
15312v2 (3.00053, 3.0006, 3.00064)
15312v3 (3.00061, 3.00065, 3.00072)
15312v4 (3.00066, 3.00074, 3.0008)
18016n0 (0.000558425, 0.000561741, 3.00081)
18016n1 (0.000562136, 0.000571402, 3.00083)
18016n2 (0.000571528, 0.000571856, 3.00084)
18016n3 (0.000571942, 0.000572263, 3.00085)
18016n4 (0.000572346, 0.000572664, 3.00086)
```

Test trzeci sprawdza, czy wiadomości VIP są dodawane na początek kolejki, a nie na koniec. Zauważamy że pomimo że wiadomości normalnych pisarzy zostały dodane do kolejki w czasie poniżej sekundy to odczytanie ich następuje na samym końcu (najpierw czytane są nowo powstające wiadomości VIP. Oprócz tego możemy zauważyć, że wiadomości VIP zachowują kolejność mimo wsadzania ich na początek kolejki.

```
Test 4: Only vip writers, last normal. [1,0,3] queue 6
15312v0 (0.000132854, 0.000142122, 3.00018)
15312v1 (0.000144002, 0.000145566, 3.00025)
15312v2 (0.000146278, 0.000147522, 3.00032)
```

```
15312v3 (0.000148182, 0.000149434, 3.00037)
15312v4 (0.000150066, 0.00015131, 3.00041)
12608n0 (1.00012, 1.00013, 3.00041)
12608n1 (1.00013, 3.0002, 3.00041)
12608n2 (3.00021, 3.00027, 3.00042)
12608n3 (3.00027, 3.00034, 3.00042)
12608n4 (3.00034, 3.00037, 3.00043)
```

Test czwarty jeszcze raz sprawdza zachowanie kolejki przy wrzucaniu normalnych wiadomości oraz wiadomości VIP. Tym razem wiadomości są wrzucane w odwrotnej kolejności: najpierw VIP potem normalne.

```
Test 5: Mixed writers [0,0,3] queue 6
18016n0 (9.2458e-05, 9.6373e-05, 3.00083)
18016n1 (9.6811e-05, 9.7522e-05, 3.00093)
15312v0 (0.00012823, 3.00096, 3.00102)
18016n2 (9.77e-05, 9.83e-05, 3.00109)
15312v1 (3.00097, 3.00112, 3.00116)
18016n3 (9.8453e-05, 9.9023e-05, 3.00121)
15312v2 (3.00112, 3.00122, 3.00126)
18016n4 (9.9174e-05, 9.9756e-05, 3.00128)
15312v3 (3.00122, 3.00129, 3.00133)
18016n5 (9.9905e-05, 3.00087, 3.00135)
15312v4 (3.0013, 3.00138, 3.00139)
15312v5 (3.00138, 3.0014, 3.00142)
15312v6 (3.00141, 3.00143, 3.00144)
15312v7 (3.00143, 3.00145, 3.00147)
15312v8 (3.00146, 3.00149, 3.00153)
15312v9 (3.0015, 3.00154, 3.0016)
18016n6 (3.00088, 3.00105, 3.0016)
18016n7 (3.00106, 3.00117, 3.00161)
18016n8 (3.00118, 3.00126, 3.00161)
18016n9 (3.00127, 3.00134, 3.00162)
```

Ostatni test sprawdza działanie programu dla różnych typów wiadomości dodawanych jednocześnie. Ponownie możemy zauważyć, że wiadomości VIP są traktowane priorytetowo i ich czasy oczekiwania są minimalne, przeciwnie niż w przypadku normalnych wiadomości.