SOI: Laboratorium 4. - Monitory
Robert Wojtaś, 283234

Zadanie do zrealizowania

Napisz w C++ dla środowiska Linux, system kolekcjonowania krótkich wiadomości tekstowych (maks. 128 znaków, ale nie mniej niż 16 znaków). Dla przypomnienia system ma bazować na synchronizacji dostępu do zasobów wykorzystujący mechanizm monitorów.

Zadaniem budowanego systemu ma być niezawodne zbieranie od klientów wiadomości, liczba klientów może być duża, ale system musi być gotowy do obsłużenia minimum 5 klientów.

Klienci - pojedynczy pod-proces lub wątek - „wrzucają” wiadomości do systemu, oprócz samej treści wiadomości wybierają priorytet wrzucanej wiadomość (np.: 0-zwykły, 1->priorytetowy).

System może zbierać wiadomości tylko w jednym pojemnym buforze. Mechanizm wkładania nowych wiadomości do tego bufora musi uwzględniać priorytety. Wszelkie operacje na buforze powinny być optymalizowane w taki sposób by nie kopiować niepotrzebnie wiadomości, oraz czas wkładania wiadomości oraz czas wyjmowania były możliwie jak najkrótsze.

Dodatkowo dla systemu utworzony ma być pod-proces lub wątek „czytelnik” zebranych wiadomości. Jego zadaniem jest pobieranie z bufora i przedstawianie wiadomości tekstowych na konsoli tekstowej. Zakłada się, że „czytelnik” będzie pobierał wiadomości z bufora, a w buforze wiadomości będą już poukładane zarówno względem priorytetów jaki i czasu ich włożenia.

Przemyśl bardzo dokładnie metodę testowania powstałego systemu, w szczególności zwróć uwagę na pokazanie równoczesnego działania wielu procesów umieszczających wiadomości w tym z różnymi priorytetami oraz współdziałanie w tym czasie „czytelnika”.

Załóż, że program testowy będą działały automatycznie generując przez klientów fikcyjne wiadomości wyłącznie tekstowe, a „czytelnik” pokazywał je na konsoli.

Rozwiązanie

W celu realizacji zadania został napisany program dołączony w załączniku – chat.cpp.
W ramach programu uruchamiana jest zdefiniowana przez użytkownika liczba wątków, reprezentujących użytkowników uprzywilejowanych oraz zwykłych. Ponadto tworzony jest czytelnik wiadomości – reader.

- priority_user – producent wysyłający wiadomości priorytetowe
- regular_user – producent wysyłający wiadomości zwykłe
- reader – konsument odczytujący wiadomości z bufora oraz wypisywanie ich na ekranie konsoli

Program bazuje na wątkach. Nie było więc konieczności imlementowania bloków pamięci współdzielonej. Synchronizacja procesów w kontekscie zasobów współdzielonych została zrealizowana za pomocą monitora.

W programie została zdefiniowana struktura Message zawierająca informacje o wiadomości: priorytet, źródło wysłania oraz jej treść – w tym zadaniu jest to tablica znaków o długości 25 znaków. Do inicjalizacji wiadomości oraz wyboru priorytetu służy funkcja set_msg, do której przekazujemy następujące wartości: priorytet, źródło, treść.

Monitor

Monitor został zaimplementowany jako klasa Buffer dziedzicząca z klasy z pliku nagłówkowego dostarczonego przez dra Kruka. Poza procedurami wpisywania I czytania wiadomości zostały stworzone funkcje zwracające aktualną zajętość bufora oraz zwracające odpowiednie iteratory. Monitor posiada następujące pola:

- full, empty – zmienne typu warunkowego
- bufor – miejsce, do którego piszą użytkownicy oraz z którego czyta czytelnik; zrealizowany za pomocą listy jednokierunkowej przy użyciu standardowej biblioteki forward_list.

- next_pri – iterator wskazujący na ostatnią wiadomość priorytetową w buforze. Element konieczny do realizacji zachowania odpowiedniej kolejności w przypadku wiadomości priorytetowych. Jego wartość uzyskiwana jest po wykonaniu funkcji find_next_pri().
- last – iterator wskazujący na ostatni element, po którym wpiszemy wiadomość zwykłą. Jego wartość uzyskiwana jest po wykonaniu funkcji find_last().
- CAPACITY – pojemność bufora, wykorzystywana przy sprawdzaniu warunków implikujących operacje na zmiennych warunkowych.

Funkcje send_msg oraz read_msg służą do wysyłania I odczytywania wiadomości I są one wykorzystywane odpowiednio przez producentów I klienta.

- Wysyłanie – jeśli wysłana wiadomość posiada priorytet równy 1 trafia w miejsce za ostatnią priorytetową wiadomością. Gdy priorytet jest równy 0 wiadomość dopisywana jest na końcu bufora.
- Czytanie – funkcja czytająca zwraca wiadomość z początku listy po czym usuwa pierwszy element.

Testy

Program chat.cpp wykonuje się po wpisaniu komendy ./chat oraz pięciu argumentów.

1. ilość użytkowników priorytetowych,
2. ilość wiadomości priorytetowych,
3. ilość użytkowników zwykłych
4. ilość wiadomości zwykłych
5. spodziewana liczba wiadomości odczytanych przez czytelnika.

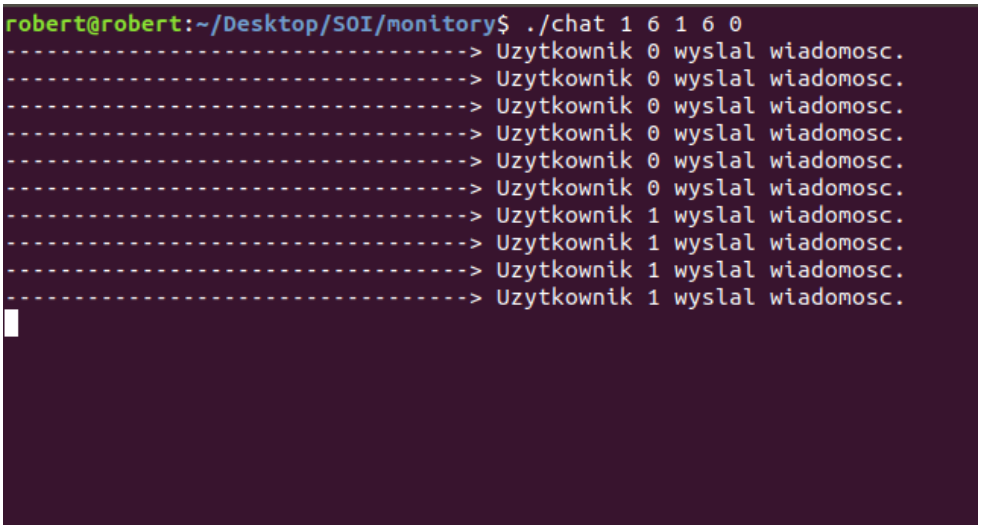
Aby zapobiec uruchamianiu wybranego procesu należy wpisać zero w miejsce argumentów 1, 3 lub 5. W celu sprawdzenia prawidłowego działania programu należało wykonać kilka testów dla różnych przypadków. Wielkość bufora została ustawiona na 10 wiadomości.

1. Brak czytania z pustego bufora

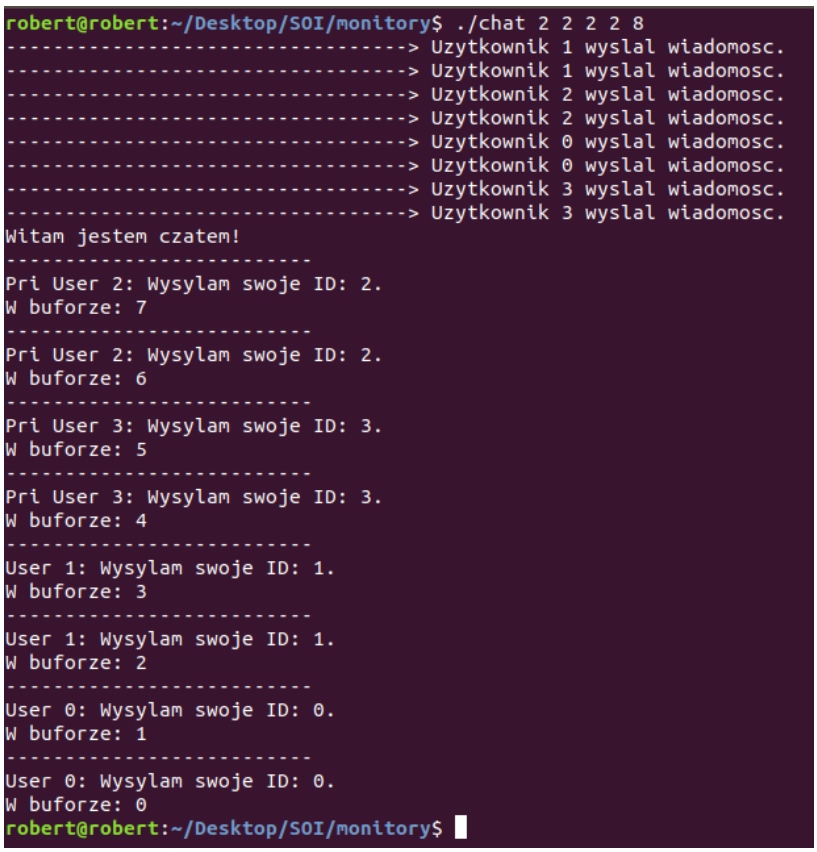
Wyłączenie producentów, czyli brak wiadomości wysłanych powoduje zawieszenie czytelnika od razu po jego uruchomieniu co pokazuje, że nie jest możliwe czytanie z pustego bufora.

2. Brak pisania do pełnego bufora

Test realizowany jest poprzez wyłączenie czytelnika oraz ustawienie liczby wiadomości na przekraczającą pojemność bufora. Ponieważ czytelnik nie działa, możliwe jest jednynie wysłanie liczby wiadomości odpowiadającej pojemności bufora. W momencie zapełnienia bufora program zawiesza się ponieważ procesy nie mogą się zakończyć. Zawieszenie programu obserwujemy to na powyższym obrazku. Można wywnioskować, że blokada pisania do pełnego bufora została zrealizowana poprawnie.

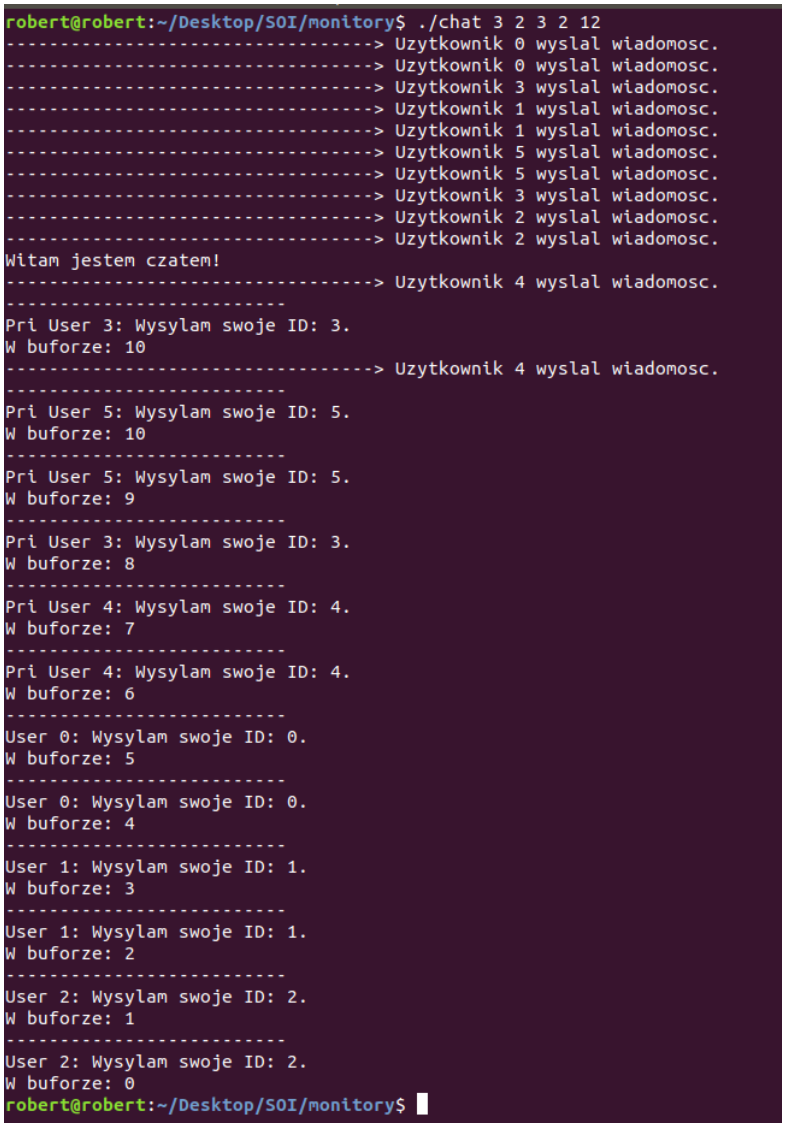


3. Uwzględnienie priorytetów



Ze względu na uruchamianie procesów w kolejności regular_user→ priority_user→ reader można łatwo zrealizować test poprawnego uwzględnienia priorytetów poprzez wysłanie kilku zwykłych wiadomości przez użytkowników zwykłych, następnie wysłanie wiadomości priorytetowych, a na końcu uruchomienie czytelnika. Po wykonaniu programu możemy zauważyć, że z bufora, w którym znajdują się wiadomości o różnych priorytetach jako pierwsze odczytywane są wiadomości priorytetowe więc zostało to zrealizowane poprawnie. Co więcej możemy zaobserwować, że kolejność odczytywania wiadomości priorytetowych jest zgodna z założeniami zadania tzn. Bufor uwzględnia kolejność nadsyłania wiadomości.

4. Przynajmniej 5 użytkowników



Na powyższym obrazku możemy zaobserwować, że uruchomienie większej ilości użytkowników nie stanowi problemu. Program działa prawidłowo, uwzględnia priorytety i wypisuje wiadomości w poprawnej kolejności.

5. Brak zagłodzeń

Test zrealizowany przez sztuczne zapełnienie bufora (utworzone dwa oddzielne wątki) wiadomościami pół na pół (priorytetowe:zwykłe). Następnie zostały uruchomione dwa wątki po jednym każdego rodzaju, które pisały do bufora oraz czytelnik. Zaobserwowano, że żaden wątek nie czeka na dostęp do bufora dłużej od pozostałych, a wiadomości są wpisywane na zmianę. Można więc wywnioskować, że nie występują zagłodzenia, zasoby są przydzielane sprawiedliwie, co świadczy o poprawnej synchronizacji.

```
robert@robert:~/Desktop/SOI/monitory$ ./chat 1 5 1 5 20
---------------------------------> Uzytkownik 100 wyslal wiadomosc.
---------------------------------> Uzytkownik 100 wyslal wiadomosc.
---------------------------------> Uzytkownik 100 wyslal wiadomosc.
---------------------------------> Uzytkownik 100 wyslal wiadomosc.
---------------------------------> Uzytkownik 100 wyslal wiadomosc.
---------------------------------> Uzytkownik 200 wyslal wiadomosc.
---------------------------------> Uzytkownik 200 wyslal wiadomosc.
---------------------------------> Uzytkownik 200 wyslal wiadomosc.
---------------------------------> Uzytkownik 200 wyslal wiadomosc.
---------------------------------> Uzytkownik 200 wyslal wiadomosc.
Witam jestem czatem!
---------------------------------> Uzytkownik 0 wyslal wiadomosc.
-------------------------
Pri User 100: Wysylam swoje ID: 100.
W buforze: 10
---------------------------------> Uzytkownik 1 wyslal wiadomosc.
-------------------------
Pri User 100: Wysylam swoje ID: 100.
W buforze: 10
---------------------------------> Uzytkownik 0 wyslal wiadomosc.
-------------------------
Pri User 100: Wysylam swoje ID: 100.
W buforze: 10
---------------------------------> Uzytkownik 1 wyslal wiadomosc.
-------------------------
Pri User 100: Wysylam swoje ID: 100.
W buforze: 10
---------------------------------> Uzytkownik 0 wyslal wiadomosc.
-------------------------
Pri User 100: Wysylam swoje ID: 100.
W buforze: 10
---------------------------------> Uzytkownik 1 wyslal wiadomosc.
-------------------------
Pri User 1: Wysylam swoje ID: 1.
W buforze: 10
---------------------------------> Uzytkownik 0 wyslal wiadomosc.
-------------------------
Pri User 1: Wysylam swoje ID: 1.
W buforze: 10
---------------------------------> Uzytkownik 1 wyslal wiadomosc.
-------------------------
Pri User 1: Wysylam swoje ID: 1.
W buforze: 10
---------------------------------> Uzytkownik 0 wyslal wiadomosc.
-------------------------
Pri User 1: Wysylam swoje ID: 1.
W buforze: 10
---------------------------------> Uzytkownik 1 wyslal wiadomosc.
-------------------------
User 200: Wysylam swoje ID: 200.
W buforze: 10
-------------------------
Pri User 1: Wysylam swoje ID: 1.
W buforze: 9
```