SOI: Laboratorium 3. - Semafory

Robert Wojtaś, 283234

Zadanie do zrealizowania

Napisz usługę „chat” dającą użytkownikom możliwość komunikacji między sobą (w obrębie jednej maszyny). Załóż, że niektórzy wśród użytkowników mają podwyższone prawa i ich wiadomości „wskakują” na początek kolejki rozsyłania wiadomości do pozostałych użytkowników. Załóż, że wszyscy użytkownicy komunikują się poprzez system w wyłącznie jednym temacie. Przemyśl metodę testowania powstałego systemu, w szczególności zwróć uwagę na pokazanie w działaniu równoczesnego „chatownia” obu grup użytkowników – w tym udowodnienia, że system faktycznie obsługuje wiadomości uwzględniając priorytety.

Rozwiązanie

W celu realizacji zadania został napisany program dołączony w załączniku – czacik.c.

W ramach programu uruchamiane są trzy procesy potomne: user_A, user_B oraz reader, których działanie przedstawiono poniżej:

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

W programie skorzystano z pamięci współdzielonej oraz trzech semaforów: full (0), empty (BUFFER_SIZE), mutex (1), których funkcje to odpowiednio: zliczanie ilości zajętych miejsc w buforze, zliczanie miejsc wolnych, realizacja wzajemnego wykluczania. Semafory inicjowane są wartościami podanymi w nawiasach. Podnoszenie I opuszczanie semaforów realizowane jest za pomocą funkcji up(int, int) I down(int, int), zrealizowanych zgodnie z opisem zawartym w dokumencie “Interprocess Communication” polecanym na stronie przedmiotu.

```
void down(int semid, int semnum)
{
   buf.sem_num = semnum;
   buf.sem_op = -1;
   buf.sem_flg = 0;
   if (semop(semid, &buf, 1) == -1)
   {
      perror("Opuszczenie semafora");
      exit(1);
   }
}

void up(int semid, int semnum)
{
   buf.sem_num = semnum;
   buf.sem_op = 1;
   buf.sem_flg = 0;
   if (semop(semid, &buf, 1) == -1)
   {
      perror("Podnoszenie semafora");
      exit(1);
   }
}
```

Programie zostały zdefiniowane struktury Message I Queue.

- **Message** – zawiera informacje o wiadomości: priorytet, źródło wysłania oraz jej treść – w tym zadaniu jest to jeden znak
- **Queue** – kolejka zawierająca tablicę wiadomości oraz wartości head, tail, size przedstawiającymi indeksy początku I końca kolejki oraz jej rozmiar.

Istnieje funkcja realizująca inicjalizację kolejki wartościami zerowymi (head, tail, size) oraz funkcja ustawiająca ustawiająca wiadomość przekazanymi wartościami (priorytet, źródło, treść).

Kolejka utworzona w funkcji main z tablicą o rozmiarze zdefiniowanej wartości BUFFER_SIZE jest faktyczną realizacją bufora komunikacyjnego, do której będziemy pisać I z której będziemy czytać.

Funkcje send_msg oraz read_msg służą do wysyłania I odczytywania wiadomości I są one wykorzystywane odpowiednio przez producentów I klienta. Bufor komunikacyjny zrealizowany jest jako kolejka cykliczna.

- **Wysyłanie** – jeśli wysłana wiadomość posiada priorytet równy 1 trafia na początek kolejki, a wartość head jest odpowiednio zmieniana. Gdy priorytet jest równy 0 wiadomość dopisywana jest w pierwszym wolnym miejscu na końcu kolejki
- **Czytanie** – proces czytający odczytuje wiadomości z elementu tablicy o indeksie head, po czym odpowiednio zmienia ten indeks

Tworzenie pamięci współdzielonej realizowana jest za pomocą funkcji shmget. Blok pamięci tworzony jest w w funkcji main poprzez podanie unikalnego klucza oraz rozmiaru bufora. Wszystkie procesy potomne proszą o dostęp do bloku również poprzez funkcję shmget. Funkcja shmat służy do przyłączania bloku pamięci do procesu I zwraca wskaźnik na utworzony bufor.

Semafory tworzone są w analogiczny sposób poprzez użycie funkcji semget.

Testy

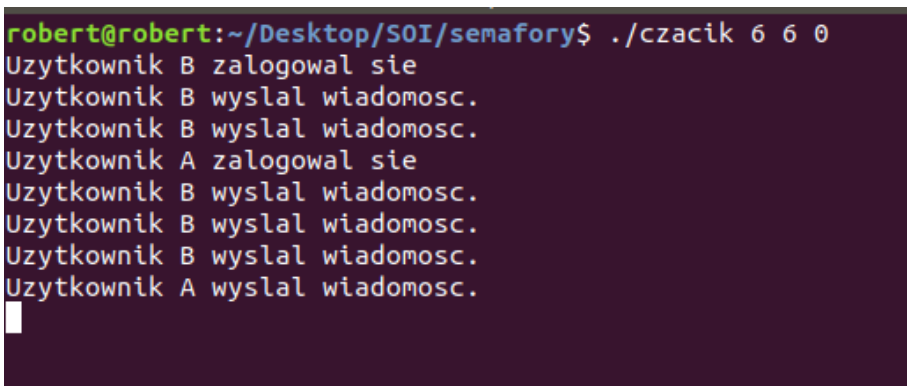
Program czacik.c wykonuje się po wpisaniu komendy czacik oraz trzech argumentów. Pierwszy z nich oznacza ilość wiadomości priorytetowych wysyłanych przez producenta A, drugi to ilość wiadomości zwykłych wysyłanych przez producenta B, trzeci jest spodziewaną liczbą wiadomości odczytanych przez czytelnika. Na konsoli możemy zaobserwować fakt uruchomienia procesów użytkowników oraz czytelnik. Sygnalizowany jest fakt wysłania wiadomości przez użytkowników, a czytelnik wyświetla komunikat o otrzymanej wiadomości I wyświetla ją w konsoli. Co więcej za sprawą czytelnika dowiadujemy się o autorze wiadomości (sprawdzenie wartości source odczytanej wiadomości). Wiadomościami są kolejne litery z imienia Robert, dla producenta A pisane wielkimi literami, dla B – małymi.

Aby zapobiec uruchamianiu wybranego procesu W celu sprawdzenia prawidłowego działania programu należało wykonać kilka testów dla różnych przypadków. Wielkość bufora została ustawiona na 6 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

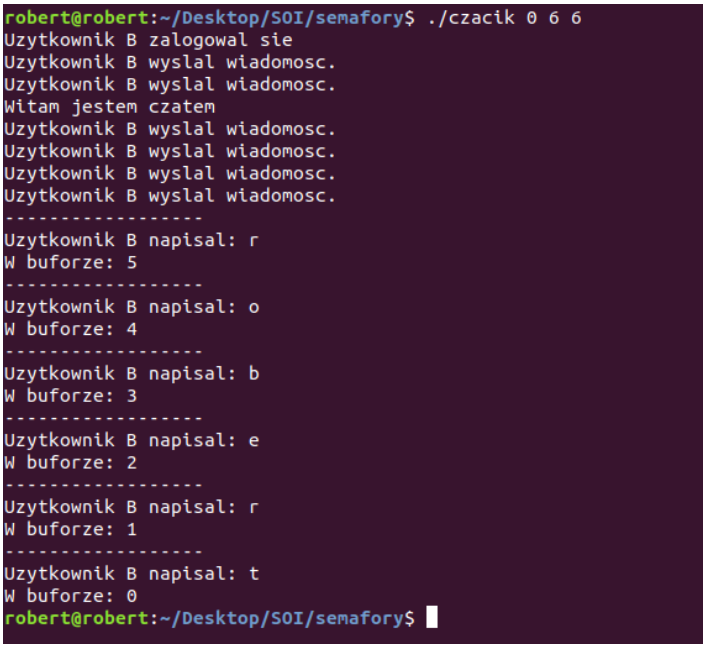
```
robert@robert:~/Desktop/SOI/semafory$ ./czacik 3 3 6
Uzytkownik B zalogowal sie
Uzytkownik B wyslal wiadomosc.
Uzytkownik A zalogowal sie
Uzytkownik B wyslal wiadomosc.
Uzytkownik B wyslal wiadomosc.
Uzytkownik A wyslal wiadomosc.
Uzytkownik A wyslal wiadomosc.
Uzytkownik A wyslal wiadomosc.
Witam jestem czatem
-----------------
Uzytkownik A napisal: B
W buforze: 5
-----------------
Uzytkownik A napisal: O
W buforze: 4
-----------------
Uzytkownik A napisal: R
W buforze: 3
-----------------
Uzytkownik B napisal: r
W buforze: 2
-----------------
Uzytkownik B napisal: o
W buforze: 1
-----------------
Uzytkownik B napisal: b
W buforze: 0
robert@robert:~/Desktop/SOI/semafory$
```

Ze względu na uruchamianie procesów w kolejności user_B → user_A → reader można łatwo zrealizować test poprawnego uwzględnienia priorytetów poprzez wysłanie kilku zwykłych wiadomości przez użytkownika B, 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 różni się od kolejności wysyłania zwykłych wiadomości.

- **Wiadomości priorytetowe** – odczytywane w kolejności od ostatniej wysłanej do pierwszej

```
robert@robert:~/Desktop/SOI/semafory$ ./czacik 6 0 6
Uzytkownik A zalogowal sie
Uzytkownik A wyslal wiadomosc.
Uzytkownik A wyslal wiadomosc.
Uzytkownik A wyslal wiadomosc.
Witam jestem czatem
Uzytkownik A wyslal wiadomosc.
Uzytkownik A wyslal wiadomosc.
Uzytkownik A wyslal wiadomosc.
-----------------
Uzytkownik A napisal: T
W buforze: 5
-----------------
Uzytkownik A napisal: R
W buforze: 4
-----------------
Uzytkownik A napisal: E
W buforze: 3
-----------------
Uzytkownik A napisal: B
W buforze: 2
-----------------
Uzytkownik A napisal: O
W buforze: 1
-----------------
Uzytkownik A napisal: R
W buforze: 0
robert@robert:~/Desktop/SOI/semafory$
```

- **Wiadomości zwykłe** – odczytywane od pierwszej do ostatniej



Wynika to ze sposobu dodawania wiadomości do kolejki – kolejne wiadomości priorytetowe “wskakują” na początek kolejki, podczas gdy zwykłe wiadomości dopisywane są na końcu kolejki.