Systemy operacyjne

Jakub Robaczewski

Implementacja:

Kolejka:

Kolejka została zaimplementowana jako struktura Queue, posiada 4 pola: length (długość), head (wskaźnik na kolejkę), usr (wskaźnik na koniec wiadomości normalnych użytkowników) oraz vip (wskaźnik na koniec wiadomości vip).

Semafory:

Semafory zostały zaimplementowane jako 3 obiekty klasy sem_t, zawierającej się w bibliotece <semaphores.h>

- QueueLock – blokada kolejki
- QueueFreeSpace – ile miejsca zostało w kolejce
- QueueNotEmpty – kolejka niepusta (zawiera wiadomości)

Są inicjowane funkcją InitSemaphores oraz usuwane funkcją RemoveSemaphores.

Budowa semafora:

```
Struct {
    int count;
    queue_t queue;
}
```

```
void down(semaphore s) {
    s.count--;
    if(s.count<0) {
        wstaw proces do s.queue;
        zablokuj proces;
    }
}
```

```
void up(semaphore s) {
    s.count++;
    if(s.count<=0) {
        usuń jeden z s.queue;
        odblokuj proces;
    }
}
```

Semafory z biblioteki <semaphore.h> realizują ogólne założenia idei semafora. Są dodatkowo rozszerzone pod względem systemu obsługi błędów.

Klasy realizowane przez wątki (Writer, WriterVIP, Reader)

Realizowane są za pomocą semaforów

- Writer
  - Czekaj na QueueFreeSpace, opuść
  - Czekaj na QueueLock, opuść
  - Wstaw wiadomość do kolejki
  - Podnieś QueueLock
  - Sprawdź czy można podnieść QueueNotEmpty
  - Zwiększ iterator
- WriterVIP
  - Czekaj na QueueFreeSpace, opuść
  - Czekaj na QueueLock, opuść
  - Wstaw wiadomość VIP do kolejki
  - Podnieś QueueLock
  - Sprawdź czy można podnieść QueueNotEmpty
  - Zwiększ iterator
- Reader
  - Czekaj na QueueNotEmpty, opuść
  - Czekaj na QueueLock, opuść
  - Pobierz pierwszą wiadomość
  - Podnieś QueueLock
  - Sprawdź czy można podnieść QueueFreeSpace
  - Zwiększ iterator

Wątki są realizowane za pomocą biblioteki <pthread.h>

Testowanie:

Program jest testowany za pomocą programu main. Składnia polecenia jest zaprezentowana poniżej:

```
./main <długość kolejki/bufora> <ilość pisarzy> <ilość pisarzy VIP> <maksymalna długość czytanej wiadomości> <ilość iteracji pisarzy>
```

Wynikiem wywołania jest log, który informuje o kolejnych zapisach i pobraniach z kolejki wiadomości, informują też o aktualnym stanie kolejki.

Możemy zauważyć w tym kilka ciekawych zdarzeń, które dowodzą poprawności programu.

```
[Writer 638400] 638400:x0
[Writer 638400] Elements in queue: 1
[VWriter 245696] 245696:v0
[VWriter 245696] Elements in queue: 2
[Reader] Reading 245696:v0
[Reader] Elements in queue: 1
```

Nawet jeśli pisarz VIP pisał po pisarzu normalnym, to jego wiadomość odczytywana jest pierwsza.

```
[Writer 758592] 758592:x3
[Writer 758592] Elements in queue: 4
[VWriter 365888] 365888:v3
[VWriter 365888] Elements in queue: 5
[Reader] Reading 3658
[Reader] Elements in queue: 4
```

Wybranie maksymalnej długości, która jest mniejsza niż długość wiadomości, obcina je do wskazanej długości.

```
[Writer 758592] 758592:x0
[Writer 758592] Elements in queue: 1
[VWriter 365888] 365888:v0
[VWriter 365888] Elements in queue: 2
[Reader] Reading 3658
[Reader] Elements in queue: 1

[Writer 758592] 758592:x1
[Writer 758592] Elements in queue: 2
[VWriter 365888] 365888:v1
[VWriter 365888] Elements in queue: 3
[Reader] Reading 3658
[Reader] Elements in queue: 2

[Writer 758592] 758592:x2
[Writer 758592] Elements in queue: 3
[VWriter 365888] 365888:v2
[VWriter 365888] Elements in queue: 4
[Reader] Reading 3658
[Reader] Elements in queue: 3
```