Operating Systems Programming Lab
Computer Engineering Department
Fall 2023/2024
Lab 6: Semaphores and Critical Sections

Objectives

1. To lean how to use semaphores for synchronization.

Prelab

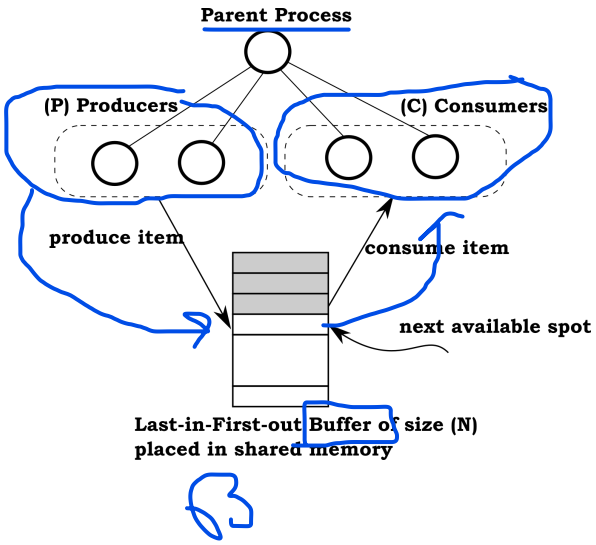
1. Read Chapter 14 of the textbook.
2. Read the manual pages of the following functions:

```
sem_t *sem_open(const char *name, int oflag);
sem_t *sem_open(const char *name, int oflag,mode_t mode, unsigned int value);
int sem_post(sem_t *sem);
int sem_wait(sem_t *sem);
int sem_close(sem_t *sem);
int sem_unlink(const char *name);
```

Experiment

You are required to implement a multi-producer multi-consumer bounded buffer system. This system is depicted in the figure below. Your program takes as input the following parameters:

- P: number of producer processes.
- C: number of consumer processes.
- N: the size of buffer (number of characters to accommodate).



The parent process works as follows:

(1) It creates three **named semaphores** as follows:

- Semaphore `full` = 0
- Semaphore `empty` = N
- Semaphore `mutex` = 1

(2) Then, it creates a shared integer variable `next`=0 (in a shared memory) as well as a shared buffer of characters of size N (in another shared memory). Characters are added/removed to the buffer in a last-in first-out manner.

(3) It forks P producers and C consumers. Producers and consumers execute the pseudocode below.

(4) Then, it waits for all children to finish. When this happens, it removes shared memories (using `shmctl`) and semaphores (using `sem_unlink`).

Producer Process

```
alarm(30); /* to terminate after 30 seconds */

do {
    /* sleep for random time [0, 4] seconds */
    ...
    /* produce a random character */
    ...
    wait(empty);
    wait(mutex);
    ...
    /* add the produced character
     to the buffer at location "next"  */

     /* increment "next"  */
    ...
    signal(mutex);
    signal(full);
} while (true);
```

Consumer Process

```
alarm(30); /* to terminate after 30 seconds */

do {
    /* sleep for random time [0, 4] seconds */
    ...
    wait(full);
    wait(mutex);
    ...
    /* remove a character from the
       buffer at location "next-1" */

    /* decrement "next"  */
    ...
    signal(mutex);
    signal(empty);
    ...
    /* print the consumed character to screen */
    ...
} while (true);
```

Sample output

```
$ ./run 3 3 20
-- [P1] Produced [0,k]
----- [C1] Consumed [0,k]
-- [P1] Produced [0,c]
----- [C1] Consumed [0,c]
-- [P2] Produced [0,s]
----- [C1] Consumed [0,s]
-- [P3] Produced [0,s]
----- [C3] Consumed [0,s]
-- [P1] Produced [0,d]
----- [C2] Consumed [0,d]
-- [P1] Produced [0,a]
----- [C3] Consumed [0,a]
-- [P1] Produced [0,x]
----- [C1] Consumed [0,x]
-- [P3] Produced [0,j]
-- [P2] Produced [1,f]
----- [C3] Consumed [1,f]
----- [C1] Consumed [0,j]
-- [P1] Produced [0,j]
-- [P1] Produced [1,s]
```