CECS 326-01 Assignment 4 (10 points)
Due: 11/9/2023 on Canvas

Recall in Assignment 3, the *master* process and its child processes executing the *slave* executable communicate through a shared memory segment. When *master* executes, it first outputs a message to identify itself, then requests to create a shared memory segment of a certain name *xxxxx*, structures it to that defined in the myShm.h header file shown below, initializes the index therein to zero, followed by creating *n* child processes. (Note that both *xxxxx* and the number *n* are obtained from the commandline parameters.) Each child process is to execute *slave*, with its child number (i.e., 1, 2, etc.) and other relevant arguments passed to it from the exec() system call. The *master* process outputs the number of *slave*s it has created, and waits for all of them to terminate. Upon receiving termination signals from all child processes, *master* then outputs content of the shared memory segment, removes the shared memory and then exits.

```
/* myShm.h */
/* Header file to be used with master.c and slave.c
*/

struct CLASS {
   int index;          /* index to next available response slot */
   int response[10];   /* each child writes its child number & lucky number here*/
};
```

From Assignment 3 you may have observed the potential problem of race condition due to the concurrent access and modification of the index variable in the shared data structure. Hence mutual exclusion will be needed for the code section where index is accessed. In this assignment, you are to use semaphore to enforce the necessary mutual exclusion.

Two implementations of semaphore are commonly available on most distributions of UNIX and Linux operating systems: System V and POSIX. In this assignment you will use the POSIX implementation. The POSIX implementation supports named and unnamed semaphores, both of which are defined in <**semaphore.h**>. The named semaphore mechanism includes **sem_wait()**, **sem_post()**, **sem_open()**, **sem_close()** & **sem_unlink()**, and should be used in this assignment. Details on the definition of these system calls and their use may be found on Linux man pages. A sample program that shows the use of these system calls can be found in the observer.c file on Canvas.

Suppose program execution is launched as follows. Notice that the specifications of the master and slave processes as described below have minor differences from those given in Assignment 3. One reason is to add semaphore and the other to highlight the potential problem due to potential race condition on the access to the monitor as an output device and the keyboard as input device.

./master n /shm_name /sem_name

master process should produce the following sequence of output:

Master begins execution
Master created a shared memory segment named */shm_name* [/shm_name is from comandline]
Master initializes index in the shared structure to zero
Master created a semaphore named */sem_name* [/sem_name is from comandline]
Master created *n* child processes to execute *slave* [The number n is from commandline]
Master waits for all child processes to terminate
Master received termination signals from all *n* child processes
Updated content of shared memory segment after access by child processes:
--- content of shared memory --- [Current content of shared memory]
Master removed the semaphore
Master closed access to shared memory, removed shared memory segment, and is exiting

slave process should produce the following sequence of output and acquire some input from user:

Slave begins execution
I am child number *x*, received shared memory name */shm_name* and */sem_name*
[*x*, */shm_name* & */sem_name* are from exec()]
Child number *x*: What is my lucky number? [Child *x* prompts user to input a lucky number]
xxx [User inputs a number in response to the above prompt]
I have written my child number to slot *y* and my lucky number to slot *y*+1, and updated index to *y*+2.
[*y* is the value in index at time of access]
Child *x* closed access to shared memory and terminates. [Same *x* as above]

Note:

1. Header files required for using POSIX semaphores and how to compile these programs may be found in sample codes observer.c.
2. Display of the above sets of output may interleave.
3. Each child process writes in 2 slots of the response array.

Your programs must run successfully on Linux.

Submit on Convas the following:

1. Two source programs *master.c* and *slave.c* and the header file `myShm.h`.
2. A screenshot that shows successful compilation of *slave.c* to *slave*, and *master.c* to *master*;
3. A screenshot that shows successful run of *master*, which will create child processes to execute *slave*; producing outputs as described above;
4. A cover page that provides your name, your student ID, course # and section, assignment #, due date, submission date, and a clear program description detailing what the programs are about. Format of the cover page should follow the cover page template posted on Canvas.
5. The programs must be properly formatted and adequately commented to enhance readability and understanding. Detailed documentation on all system calls are especially needed.