CECS 326-01 Assignment 3 (10 points)

Due: 10/19/2023 on Canvas

In this assignment you will make use of the POSIX implementation of the Linux shared memory mechanism, and use the fork(), exec() and wait() system calls to create child processes and control their execution. For this assignment you need to write two C programs named *master.c* and *slave.c*, which should be compiled into executables *master* and *slave*, respectively. Together they should do the following:

When *master* executes, it should first output a message to identify itself. It should then request to create a shared memory segment of a certain name *xxxxx*, structure it to that defined in the myShm.h header file shown below, initialize 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 the shared memory segment name *xxxxx* passed to it from the exec() system call. The *master* process should output the number of *slave*s it has created, and wait for all of them to terminate. Upon receiving termination signals from all child processes, *master* then outputs the entire content of the shared memory segment, removes the shared memory and then exits.

The following header file that defines a C struct should be used to structure the shared memory segment:

/\* 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 here

};

Suppose program execution is launched as follows:

./master 3 my\_shm\_name

*master* should produce the following sequence of output:

Master begins execution

Master created a shared memory segment named *my\_shm\_name*  [ my\_shm\_name is from comandline ]

Master created *3* child processes to execute *slave*  [ The number 3 is from commandline]

Master waits for all child processes to terminate

Master received termination signals from all *3* child processes

Updated content of shared memory segment after access by child processes:

--- content of shared memory --- [ Current content of shared memory ]

Master removed shared memory segment, and is exiting

When a child process executes *slave*, it should first output a message to identify itself, and show its child number and the shared memory segment name it obtained from the exec() system call that invokes its execution. It should then open the existing shared memory segment, acquire access to it, and write its child number into the next available slot in the shared memory structure, close the shared memory and terminate.

*slave* should produce the following sequence of output:

Slave begins execution

I am child number *x*, received shared memory name *my\_shm\_name*. [ *x* and *my\_shm\_name* from exec() ]

I have written my child number to slot *y* and updated index to *y*+1. [*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. Required header files and how to compile these programs may be found in sample codes given in Lecture Notes.
2. Display of the above sets of output may interleave.
3. Suppose 3 child processes are created, the child numbers 1, 2, 3 written to the response array may not appear in this order.
4. In some extreme cases due to race condition, the number of child numbers written to the response array may be smaller than the number of child processes that have accessed it.

The program must run successfully on Linux.

**Submit on Convas** the following:

1. The source programs *master.c* and *slave.c*;
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.