CECS 326-01 Assignment 4 (10 points)
Due: 11/3/2022 on BeachBoard

Recall in Assignment 3, the *master* process and its child processes executing the *slave* executable communicate through a shared memory segment that is structured as struct CLASS. The correct behavior of their execution should be as follows:

master process: {NOTE: *master* produces same outputs as described in Assignment 3.}

master obtains shared memory segment name (say, *yyyyy)* and number of child from commandline.
master requests to create a shared memory segment, and structures it according to struct CLASS.
master initializes index in the shared structure to zero.
master creates *n* child processes (*n* from commandline), and has every one to execute the *slave* executable.
master waits for all *n* child processes to terminate.
master displays content of the shared structure.
master closes access to shared memory and removes the shared memory segment, then exits.

slave process: {NOTE: Below describes how *slave* in Assignment 3 is supposed to behave. *slave* outputs are slightly different from Assignment 3.}

slave outputs the following:
Slave begins execution
I am child number *x*, received shared memory name *yyyyy*
slave acquires access to shared memory segment, and structures it according to struct CLASS.
slave copies index to a local variable i.
slave writes its child number in response[index].
slave increments index.
slave closes access to shared memory segment.
slave outputs the following:
I have written my child number *x* to response[i] in shared memory
Slave closed access to shared memory segment and terminates
slave exits.

{NOTE: If your Assignment 3 implementation does not work as above, you need to modify it.}

Based on the above description, you can see 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 BeachBoard.

Your programs must run successfully on Linux.

Do the following for this assignment:

1. Add necessary synchronization code in your Assignment-3 C programs to correct potential problems due to race condition, and compile them into executables *master* and *slave*, respectively. Make sure that sufficient and proper comments are included on the added code as well as the existing code.
2. Run your corrected version of *master* (with *slave*) to make sure that the programs behave as required and the outputs are correct.
3. Submit on BeachBoard the two corrected programs, along with the *myShm.h* file, a screenshot that shows successful compile of both programs as well as a successful run, and 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 you have done for the correction. Format of the cover page should follow the cover page template on BeachBoard.

4. The programs must be properly formatted and adequately commented to enhance readability and understanding. Detailed documentation on all system calls are especially needed.