Operating System Lab

1. You will implement system calls to get information about currently active processes, much like the ps and top commands in Linux do. Implement the system call **getNumProc(),** to return the total number of active processes in the system (either in embryo, running, runnable, sleeping, or zombie states). Also implement the system call **getMax-Pid()** that returns the maximum PID amongst the PIDs of all currently active (i.e., occupying a slot in the process table) processes in the system.
2. Copy **read-write4.c**. Compile it with gcc and run it. Answer the following:
   i. This program contains a third semaphore, mutex. Explain the purpose of this semaphore
   ii. Program **`read-write-4.c`** prevents any reader from working at the same time as any writer. Assuming that the buffer contains several locations, however, writing to one buffer location should not interfere with reading from another. That is, the critical section for readers need not be considered exactly the same as the critical section for writers. Remove semaphore `mutex` and add additional semaphores, as needed, so that some reader could work concurrently with some writer (assuming the buffer contained some data but was not full -- so both reading and writing made sense).
3. Copy **read-write-3.c.** Compile it with gcc and run it. With this use of semaphores, explain whether your code in step 11 will work when there are multiple readers or when there are multiple writers. In each case, if the code would work, explain why. If the code would not work, give a timing sequence involving the several readers and/or writers showing what might go wrong.