Semaphores and Message Passing

The goal of this homework is to become familiar with semaphores in the Unix operating system. Do the same project as Assignment 2, except that now, you use a monitor to control access to critical resources.

You can get a good feel by first architecting the project in a way to be solved by monitor. Towards that effect, create pseudocode that will allow you to execute the critical section problem independently, that is, without the entry and exit section. This function will be passed to the monitor which will ensure that only one process is actually executing inside its critical section (or inside the monitor).

After that, you figure out how to implement the monitor, this time using the semaphores as illustrated in the notes, and do it as a library. This library will take any function to execute but ensure that only one process will execute inside the monitor.

Make sure that you have signal handing to terminate all processes, if needed. In case of abnormal termination, make sure to remove shared memory and semaphores as well. Use `semget(2)`, `semctl(2)`, and `semop(2)` to implement monitor.

Your monitor should receive the names of functions passed as parameters. Since different functions are required to access the critical resource, passing the functions as parameters will prove a clean option.

What to handin

Hand in an electronic copy of all the sources, README, Makefile(s), and results. Create your programs in a directory called `username.3` where `username` is your login name on hoare. Once you are done with everything, remove the executables and object files, and issue the following commands:

```
% cd
% ~sanjiv/bin/handin cs4760 3
```

Do not forget Makefile (with suffix rules), RCS, and README for the assignment. If you do not use RCS, you will lose 10 points. Omission of a Makefile (with suffix rules) will result in a loss of another 10 points, while README will cost you 5 points. Run the problem for five minutes by default and kill everything at that point if it is not completed. Also print messages when a process waits for and acquires a semaphore using a logfile just as in the last assignment.