CS 4760 **Operating Systems**
Assignment # 3 Due Date: October 11, 2017

Semaphores and Operating System Simulator

The goal of this homework is to become familiar with semaphores in the Unix operating system. In this project you will create an empty shell of an OS simulator and do some basic tasks in preparation for a more comprehensive simulation later. This will require the use of `fork`, `exec`, shared memory, and semaphores.

Operating System Simulator

This will be your main program and serve as the master process. You will start the operating system simulator (call the executable `oss`) as one main process which will start by forking off a number of processes. Later, it will fork off new children as some terminate.

As we create the simulator, we'll maintain our own system clock. `oss` will start by first allocating shared memory for a clock that only it will increment (write/modify). The child processes should be able to view this memory but will not increment it (readonly). This shared memory clock should be two integers, with one integer to hold seconds and the other integer to hold nanoseconds. If one integer has 5 and the other has 10000 then that would mean that the clock is showing 5 seconds and 10000 nanoseconds. This clock should be initialized to 0 by `oss`.

In addition to the clock in shared memory, there should be an additional area of shared memory allocated to allow the child processes to send information to `oss`. Let us call this area `shm_msg`.

With the clock at zero, `oss` should fork off the appropriate number of child processes to start. Then it should enter into a loop. Every iteration of that loop, it should increment the clock (simulating time passing in our system). While in this loop, `oss` checks `shm_msg`. If it has been sent a message from a child, it should output the contents of that message (which should be a set of two integers, a value in our clock) to a file. If there is a message in `shm_msg`, that means that the child process is terminating, `oss` should then fork off another child and clear `shm_msg`. This process should continue until 2 seconds have passed in the simulated system time, 100 processes in total have been generated, or the executable has been running for the maximum time allotted (in real seconds). At that point, the master should terminate all children and then itself, while making sure to clean up any shared memory and semaphores it is using.

Note that I did not specify the amount of time that `oss` should increment the clock by on each iteration. This will depend on your implementation. Tune it so that your system has some turnover.

The log file should at least have the following information, but can contain additional information:

```
Master: Child pid is terminating at time xx.yy because it reached mm.nn in slave
Master: Child pid is terminating at time xx.yy because it reached mm.nn in slave
Master: Child pid is terminating at time xx.yy because it reached mm.nn in slave
...
```

with `xx.yy` being the time in the simulated system clock when `oss` received the message and `mm.nn` is the time that the user process put in `shm_msg`.

User Processes

The child processes of the `oss` are the user processes. These should be a separate executable from master, run using an `exec` from the fork of `oss`. The excutable should be named `child`.

`child` should start by reading the simulated time system clock generated by `oss`. It should then generate a random duration number from 1 to 1,000,000 and add this to the time it got from the clock. This represents the time when `child` should terminate itself. `child` should then loop continually over a critical section of code. This critical section should be enforced through the use of semaphores.

During each iteration over the critical section, `child` should examine the `oss`-managed clock and see if that duration has passed. If, while in the critical section, `child` sees that its duration is up and there is nothing already in `shm_msg`, it should send a message to `oss` that it is going to terminate. Once it has put a message in `shm_msg`, it should terminate itself, making sure to cede the critical section to any other user processes before doing so. If `shm_msg` is not empty, `child` should cede the critical section and then, on the next time it enters the critical section, try and terminate at that time.

This checking of duration vs `oss` clock and putting a message in the `shm_msg` for master should only occur in the critical section. If a user process gets inside the critical section and sees that its duration has not passed, it should cede the critical section to someone else and attempt to get back in the critical section.

Note: Make sure that you have signal handing to terminate all processes, if needed. In case of abnormal termination, make sure to remove any resources that are used, including shared memory and semaphores.

Your main executable should use command line arguments. You must implement at least the following command line options using `getopt`:

```
-h
-s x
-l filename
-t z
```

where `x` is the maximum number of slave processes spawned (default 5) and `filename` is the log file used. The parameter `z` is the time in seconds when the master will terminate itself and all children (default 20).

Implementation

The code for `oss` and `child` processes should be compiled separately and the executables be called `oss` and `child`. The program should be executed by

```
./oss [-s x] [-l filename] [-t z]
```

where the parameters inside square brackets are optional.

Hints

I highly suggest you do this project incrementally. Test out the command line options, then spawn the slave processes but just have them all terminate. Then encode the shared memory and termination after a specified time. Then insert semaphores and enforcement of critical region. Then check to see if the child processes are able to communicate with master. Lastly try and get `oss` to spawn new children as others terminate.

What to handin

Handin 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 revision control, you will lose 10 points. I want to see a log of how the project got modified. Omission of a `Makefile` (with suffix rules) will result in a loss of another 10 points, while `README` will cost you 5 points. Make sure that there is no IPC structures left after your process terminates (normal or interrupted termination). Also, use relative path to execute the child. Run the problem for prescribed time 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.