CS4760 Operating Systems - Project No. 3

Due Date April 2, 2017 Max. Points : 100

Deliver your project using Mygateway to 23.59 on the due date.

Important 1: Please do all assignments on hoare

Important 2: Please remember that you need to present me your program during office hours after submitting it using Mygateway.

Message Passing and Operating System Simulator

In this project you will be creating an empty shell of an OS simulator and doing some very basic tasks in preparation for a more comprehensive simulation later. This will require shared memory and some message passing.

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 who will fork multiple children and then fork off new children as some terminate.

oss will start by first allocating shared memory for a clock that only it will increment. The child processes should be able to view this memory but will not increment it. This shared memory clock should be two integers. One integer to hold seconds, the other integer to hold nanoseconds. So, 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 start at 0.

With the clock at zero, oss should then fork off the appropriate number of child processes. Then it should enter into a loop. Every iteration of that loop, it should increment the clock (**simulating time passing in our system – virtual clock**). While in this loop, oss should be waiting for messages from its children. When sent a message, it should output the contents of that message (which should be a set of two integers, a value in our clock) to a file. The master should then fork off another child. This process should continue until 2 seconds have passed in the simulated system, 100 processes in total, have been generated or the executable has been running for the maximum time allotted. At that point the master should terminate all children and then itself.

Note that I did not specify how much oss should increment the clock on each iteration. This will depend on your implementation. Tune it so that your system has some turnover. The log file should look as follows:

Master: Child pid is terminating at my time xx.yy because it reached mm.nn in slave
Master: Child pid is terminating at my time xx.yy because it reached mm.nn in slave
Master: Child pid is terminating at my 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 slave sent the message to master.

User Processes

The child processes of the oss are the **user processes**. These should be a separate executable from master, run with exec from the fork of oss.

This process should start by reading the system lock generated by oss. It should then generate a random duration number from 1 to 100000. This represents how long this child should run.

It should then loop continually over a critical section of code. This critical section should be enforced through the use of message passing and msgsnd and msgrcv.

In this critical section of code done using message passing, the user process should read the oss clock until that duration has passed. If while in the critical section it sees that its duration is up, it should send a message to oss that it is going to terminate. Once the child knows that master received the message, it should terminate itself. The message sent to oss should consist of the current oss system clock time that it decided to terminate on.

This checking of duration vs oss clock and messaging to 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.

Your main executable should use command line arguments. You must implement at least the following command line arguments 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 user processes should be compiled separately and the executables be called oss and user. The program should be executed by ./oss

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 do message passing and enforcement of critical region. Lastly try and get master to spawn new children as others terminate.

What to handin

Handin an electronic copy of all the sources, README, Makefile(s), and results using Mygateway. Do not forget Makefile (with suffix or pattern rules), and README for the assignment. It will be nice when you will use some system for version control. This README should tell me how to run your project, and any problems or issues that it has running. Omission of a Makefile will result in a loss of 10 points, while README will cost you 5 points. Make sure that there is no shared memory left after your process finishes (normal or interrupted termination). Do not forgot include in the README file your name and the date when your project was submitted. I do not like to see any extensions on Makefile and README files.