CS4760 Operating Systems - Project No. 2

Due Date March 14, 2017 Max. Points : 100

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

Important: Please do all assignments on hoare

Concurrent UNIX Processes and shared memory

The goal of this homework is to become familiar with concurrent processing in Unix/Linux using shared memory. You will write a program that uses multiple processes to increment a number in shared memory and write a message into a file. The multiple processes compete to get exclusive access to the file to write into, participating in possibly two race conditions (on the variable in shared memory and the file access). You will use the Peterson or any other algorithm for the critical section problem. The master process should allocate an integer in shared memory (sharedNum) and initialize it to zero. Then generate by default 5 slave processes using a master process and make them write into a file called test.out in the directory where the processes are running from. The message to be written into the file is:

File modified by process number xx at time ssnn with sharedNum = yy

where xx is the process number as specified in the master and tt is the time in seconds and nanoseconds. The value of xx is between 1 and the max number of slaves (master process is taken to be process 0). This implies that the child process will be run by the command

slave xx

If a process starts to execute code to enter the critical section, it must print a message to that effect on stderr. It will be a good idea to include the time when that happens. Also, indicate the time on stderr when the process actually enters and exits the critical section. Within the critical section, wait for [0-2] seconds before you increment the variable and write into the file, and then, wait for another [0-2] seconds before leaving the critical section. Each slave should only enter its critical region by default only 3 times before it terminates. This can be changed per command line options discussed later. The code for each slave should use the following template:

```
for ( i = 0; i < max_writes; i++ )
   {
     execute code to enter critical section;
     /* Critical section */
     sleep for random amount of time (between 0 and 2 seconds);
     increment sharedNum
     write message into the file
     sleep for random amount of time (between 0 and 2 seconds);
     exit from critical section;
   }
```

Your main executable should use command line arguments. You must implement at least 3 command line arguments using getopt:

-h
-s x
-l filename
-i y
-t z

where x is the maximum number of slave processes spawned (default 5) and filename is the log file used. The parameter y determines how many times each slave should increment and write to the _le before terminating (default of 3).
The parameter z is the time in seconds when the master will terminate itself (default 20)
You will be required to create 5 separate slave processes from your master. That is, master will just spawn the 5 slaves and wait for them to finish. It will also set up the signal handler. It also sets a timer at the start of computation to 20 seconds (make it configurable). If computation has not finished by this time, master kills all the spawned processes and then exits. Make sure that you print appropriate message(s) from both master and each slave.
In addition, master should print a message when an interrupt signal (^C) is received. Make sure that all the children/grandchildren are killed by master when this happens, and all the shared memory is deallocated. The slaves kill themselves upon receiving interrupt signal but print a message on stderr to indicate that they are dying because of an interrupt, along with the identification information. Make sure that the processes handle multiple interrupts correctly. As a precaution, add this feature only after your program is well debugged.

Implementation
The code for master and slave processes should be compiled separately and the executables be called master and slave. The program should be executed by

./master

Other points to remember: You are required to use fork, execl (or one of its variants), wait, and exit to manage multiple processes. Use shmctl suite of calls for shared memory allocation. Also make sure that you do not have more than twenty processes in the system at any time. You can do this by keeping a counter in master that gets incremented by fork and decremented by wait.

Hints
You will need to set up shared memory in this project to allow the processes to communicate with each other. Please check the man pages for shmget, shmctl, shmat, and shmdt to work with shared memory. You will also need to set up signal processing and to do that, you will check on the functions for signal and abort. If you abort a process, make sure that the parent cleans up any allocated shared memory before dying.

In case you face problems, please use the shell command ipcs to find out any shared memory allocated to you and free it by using ipcrm.
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 signal handling and termination after a specified time. Then do shared memory and ensure all processes have access to it. ONLY AFTER THAT WOULD

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.