CS4760 Operating Systems - Project No. 5

Due Date April 30, 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.

Resource Management

In this part of the assignment, you will design and implement a resource management module for our Operating System Simulator **oss**. In this project, you will use the deadlock detection and recovery strategy to manage resources.
There is no scheduling in this project, but you will be using shared memory so be cognizant of possible race conditions.

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 at random times. The randomness will be simulated by a logical clock that will be updated by **oss** as well as user processes. The logical clock resides in shared memory and is accessed as a critical resource using a semaphore. You should have two unsigned integers for the clock; one will show the time in seconds and the other will show the time in nanoseconds, offset from the beginning of a second.

In the beginning, **oss** will allocate shared memory for system data structures, including resource descriptors for each resource. All the resources are static but some of them may be shared. The resource descriptor is a fixed size structure and contains information on managing the resources within **oss.** Make sure that you allocate space to keep track of activities that affect the resources, such as request, allocation, and release. The resource descriptors will reside in shared memory and will be accessible to the child. Create descriptors for 20 resources, out of which about 20% should be sharable resources. After creating the descriptors, make sure that they are populated with an initial number of resources; assign a number between 1 and 10 (inclusive) for the initial instances in each resource class. You may have to initialize another structure in the descriptor to indicate the allocation of specific instances of a resource to a process.

After the resources have been set up, fork a user process at random times (between 1 and 500 milliseconds of your logical clock). Make sure that you never have more than 18 user processes in the system. If you already have 18 processes, do not create any more until some process terminates. Your user processes execute concurrently and there is no scheduling performed. They run in a loop constantly till they have to terminate.

oss should grant resources when asked as long as it can find sufficient resources to allocate. Periodically, say every second, **oss** runs a deadlock detection algorithm. If there are some deadlocked processes, it will kill them one by one till the deadlock is resolved. If you use some policy to kill processes, put that in your README file as well as in the code documentation. Make sure to release any resources claimed by a process when it is terminated.

User Processes
While the user processes are not actually doing anything, they will be asking for resources at random times.

You should have a parameter giving a bound for when a process should request/let go of a resource. Each process when it starts should roll a random number from 0 to that bound and when it occurs it should try and either claim a new resource or release a resource that it already has. It should make this request by putting a request in shared memory. It will continue looping and checking to see if it is granted that resource.

At random times (between 0 and 250ms), the process checks if it should terminate. If so, it should deallocate all the resources allocated to it. However, just to ensure that we have some churn in our system, ensure that no process terminates itself in the system until it has been around for at least 1 second.

The statistics of interest are:

- Throughput
- Turnaround time
- Waiting time
- CPU utilization

Make sure that you have signal handling to terminate all processes, if needed. In case of abnormal termination, make sure to remove shared memory and semaphores.

When writing to the log file, you should have two ways of doing this. One setting (verbose on) should indicate in the log file every time master gives someone a requested resource or when master sees that someone has finished with a resource. It should also log the time when a deadlock is detected and how it removed the deadlock. That is, which processes were terminated.

When verbose is off, it should only log when a deadlock is detected and how it was resolved.

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 to 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.