CS 4760 **Operating Systems**
Assignment # 5 Due Date: April 15, 2020

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. Thus, 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[1]. 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. If it cannot allocate resources, the process goes in a queue waiting for the resource requested and goes to sleep. It gets awakened when the resources become available, that is whenever the resources are released by a process. 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, notice 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 ask for resources at random times. You should have a parameter giving a bound B for when a process should request (or let go of) a resource. Each process, when it starts, should generate a random number in the range $[0, B]$ and when it occurs, it should try and either claim a new resource or release an already acquired resource. It should make the request by putting a request in shared memory. It will continue to loop and check to see if it is granted that resource.

The user processes can ask for resources at random times. Make sure that the process does not ask for more than the maximum number of available resource instances at any given time, the total for a process (request + allocation) should always be less than or equal to the maximum number of instances of a specified resources.

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 by communicating to `oss` that it is releasing all those resources. Make sure to do this only after a process has

[1] *about* implies that it should be $20 \pm 5\%$ and you should generate that number with a random number generator.

run for at least 1 second. If the process is not to terminate, make the process request some resources. It will do so by putting a request in the shared memory. Also update the system clock. The process may decide to give up resources instead of asking for them.

I want you to keep track of statistics during your runs. Keep track of how many requests have been granted. as well as the number of processes that are terminated by the deadlock detection/recovery algorithm vs processes that eventually terminated successfully.lso note how many times the deadlock detection is run, how many processes it had to terminate, and percentage of processes in a deadlock that had to be terminated on an average.

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 `oss` gives someone a requested resource or when master sees that a user 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. In addition, every 20 granted requests, output a table showing the current resources allocated to each process.

An example of possible output might be:

```
Master has detected Process P0 requesting R2 at time xxx:xxx
Master granting P0 request R2 at time xxx:xxx
Master has acknowledged Process P0 releasing R2 at time xxx:xxx
Current system resources
Master running deadlock detection at time xxx:xxxx:
   Processes P3, P4, P7 deadlocked
   Attempting to resolve deadlock...
   Killing process P3:
      Resources released are as follows: R1:1, R3:1, R4:5
   Master running deadlock detection after P3 killed
   Processes P4, P7 deadlocked
   Killing process P4:
      Resources released are as follows: R1:3, R3:1, R4:2
   System is no longer in deadlock
Master has detected Process P7 requesting R3 at time xxx:xxxx
...
   R0  R1  R2  R3  ...
P0  2   1   3   4  ...
P1  0   1   1   0  ...
P2  3   1   0   0  ...
P3  7   0   1   1  ...
P4  0   0   3   2  ...
P7  1   2   0   5  ...
...
```

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

Regardless of which option is set, keep track of how many times `oss` has written to the file. If you have done 100000 lines of output to the file, stop writing any output until you have finished the run.

Note: I give you broad leeway on this project to handle notifications to `oss` and how you resolve the deadlock. Just make sure that you document what you are doing in your README.

Deliverables

Handin an electronic copy of all the sources, `README`, Makefile(s), and results. Create your programs in a directory called *username*.5 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 5
```