CS 4760 **Operating Systems**
Assignment # 5 Due Date: November 14th, 2023

Resource Management

Purpose

The goal of this homework is to learn about resource management inside an operating system. You will work on the specified strategy to manage resources and take care of any possible starvation/deadlock issues.

Task

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 the main process who will fork multiple children at specific times as in previous projects. You will take in the same command line options as the previous project.

In the beginning, `oss` will allocate shared memory for our clock. In addition, you will need a process table as we had in previous projects, but possibly with different entries. You will also need to allocate a resource table. All the resources are static and should have a fixed number of instances defined in a header file. 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. Create descriptors for 10 resources, with 20 instances of each resource. After creating the descriptors, make sure to initialize the data structures. 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` initially start by creating a child process. Make sure that you never have more than 18 user processes in the system no matter what options are given, but other than that periodically try to add a process to the system as long as it does not violate our limits. If it would violate the limits, just do not launch it and wait until the next interval to try and launch it (but do not consider it as having launched as far as the total). The workers will run in a loop constantly till they decide to terminate.

`oss` should grant resources when asked as long as it can find sufficient quantity 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, every simulated 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. The policy you use to make the determination of which one to terminate should be 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.

Important: As oss is incrementing the clock continually while checking for messages, it should do a msgrcv using `IPC_NOWAIT` as the last parameter to the msgrcv. I have an example of this available under our canvas Modules.

Overview of the procedure

```
while (stillChildrenToLaunch or childrenInSystem) {
       do a nonblocking waitpid to see if a
       child process has terminated
        if so, free up its resources

       determine if we should launch a child

       check to see if we can grant any outstanding
       requests for resources by processes
       that didnt get them in the past

       check if we have a message from a child

       if (msg from child) {
          if (request)
           grant it if we can
          if (release)
           release it
       }

       Every half a second, output the resource table and
       process table to the logfile and screen

       every second, check for deadlock

       if (deadlock)
          terminate some processes until deadlock is gone
}
```

User Processes

While the user processes are not actually grabbing resources (as we are a simulation), they will ask for resources at random times. You should have a parameter giving a bound B in nanoseconds for when a process should request (or release) a resource. Each process, every time it is scheduled, 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. This random bound B should be tuned by you low enough so processes request resources often enough so they get deadlocked. It should make the request by sending a message to oss. It should then wait to get a message back indicating it was granted before continuing on. In the case that it was blocked (ie: not given the resource), it would just be stuck waiting on a message.

When a process decides to request a resource, it should only request one instances of it and it should send a message indicating that it wants that resource. However, make sure that the process does not ask for more than the maximum number of 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 resource. This requires that each process keep track of how many resources they have of each particular type and for them to know the meximums but they DO NOT need to know what any other process wants or needs. Just to emphasize this point, a process could over time end up requesting all the resources of the system and in fact would be granted this if it was the only process in the system.

As each process could request or release resources, we should prefer that processes request resources more than they release them. This should be a parameter in your system for this. You will need to tune this so your processes are likely to deadlock. I would suggest to start this quite high. That would mean that individual processes would quickly request many resources in the system and so we should deadlock quickly. That is a good thing, as we want to test our deadlock detection algorithm.

Every 250ms, a worker process should check if it should terminate. If so, it should release 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 run for at least 1 second. If the process is not to terminate, make the process request (or release) some resources. It will do so by putting a request in a message queue. The request should never exceed the total number of resources of that class within the system. Also update the system clock.

I want you to keep track of statistics during your runs. Keep track of how many requests have been granted immediately and how many are granted after waiting for a bit. Also track the number of processes that are terminated by the deadlock detection/recovery algorithm vs processes that eventually terminated successfully. Also 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 oss sees that a user has finished with a resource. It should also log the time when a request is not granted and the process goes to sleep waiting for the resource. In addition, every 20 granted requests, output a table showing the current resources allocated to each process.

Invoking the solution

Execute `oss` with the parameters as previous projects. You may add some parameters to modify simulation, such as number and instances of resources; if you do so, please document it in your `README`.

Log Output

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 running deadlock detection at time xxx:xxxx: No deadlocks detected
Master has acknowledged Process P0 releasing R2 at time xxx:xxx
   Resources released : R2:1
Current system resources
 ...
Master has detected Process P3 requesting R4 at time xxx:xxx
Master: no instances of R4 available, P3 added to wait queue at time xxx:xxx
Master running deadlock detection at time xxx:xxxx:
   Processes P3, P4, P7 deadlocked
   Master terminating P3 to remove deadlock
   Process P3 terminated
      Resources released: R1:3, R7:4, R8:1
Master running deadlock detection at time xxx:xxxx: No deadlocks detected
Process P2 terminated
   Resources released: R1:1, R3:1, R4:5
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 indicate what resources are requested and granted, and available resources. 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.

Suggested Implementation Steps

I'll suggest that you do the project incrementally. You are free and encouraged to reuse any functions from your previous projects.

- Start by creating a `Makefile` that compiles and builds the two executables: `oss` and `user_proc`. [1 day]
- Implement clock in shared memory; possibly reuse the one from last project. [1 day]
- Have `oss` create resource descriptors and populate them with instances. [2 days]
- Create child processes; make them ask for resources and release acquired resources at random times. [4 days]
- Use shared memory or message queues to communicate requests, allocation, and release of resources. Indicate the primitive used in your `README`. [2 days]
- Implement deadlock detection and recovery algorithm. You are free to use the code that is given to you in the notes. [2 days]
- Keep track of output statistics in log file.

Feel free to ask questions about clarifications.

Termination Criterion

`oss` should stop generating processes if it has already generated up to n processes, or if more than 5 real-time seconds have passed. If you stop adding new processes, the system should eventually have no children and then, it should terminate. Tune your parameters so that the system is able to encounter a deadlock state and that is shown in log file.

Criteria for Success

Make sure that the code adheres to specifications. Document the code appropriately to show where the specs are implemented. *You must clean up after yourself.* That is, after the program terminates, whether normally or by force, there should be no shared memory, semaphore, or message queue that is left allocated to you.

Grading

1. *Overall submission: 30pts*. Program compiles and upon reading, seems to solve the assigned problem in the specified manner.
2. *README/Makefile: 10pts*. Ensure that they are present and work appropriately.
3. *Code readability: 10pts*. Code should be readable with appropriate comments. Author and date should be identified.
4. *Conformance to specifications: 50pts*. Algorithm is properly implemented and documented.

Submission

Handin an electronic copy of all the sources, `README`, Makefile(s), and results. Create your programs in a directory called *username*.6 where *username* is your login name on opsys. Once you are done with everything, *remove the executables and object files*, and issue the following commands:

```
chmod 700 username.5

cp -p -r username.5 /home/hauschildm/cs4760/assignment5
```