#### COEN 177: Operating Systems

**Lab assignment 7: Minix Scheduling**

**Objectives**

##### To understand Minix scheduler implementation.

##### To modify the Minix scheduler and observe effects.

##### 

##### **Guidelines**

In Lab6, you learned to setup a Minix OS virtual machine and how to access the Minix kernel. You have also learned how to setup and use ftp to make changes to files locally rather than on the virtual machine. This allowed you to make changes to the kernel, compile, and rebuild the kernel again. In this lab, the scheduling algorithm of Minix will be modified. Like Linux, Mac and Window, the multi-level feedback queue (MFQ) scheduler is used in Minix.

MFQ has multiple Round Robin queues, each with a different priority level and time quantum. High priority queues have short time slices (quantum). A task at a higher priority queue preempts lower priority tasks and every time the task uses up its time quantum, it drops a level every time the task yields the processor because it is waiting on I/O. It stays at the same level (or is bumped up a level) and if the task completes it leaves the system.

So, in Minix each queue is assigned a priority and making changes to high priority queues may lead to making the Minix system unstable. You will need to experiment with what changes can be done safely.

**Reminder - Getting started with Minix on the ECC Systems**

For consistency, it is recommended that you run the provided Minix image on the ECC systems. If you have not done so already, follow the instructions in Lab 6 to install and run Minix on the ECC Systems. Make sure you know how to setup and use the FTP and how to get the IP address of Minix virtual machine.

##### **The Minix Scheduler**

In this lab, you will gain additional hands-on experience to modify the Minix operating system kernel, specifically the process scheduling algorithm.

1. Locate the scheduler function on the /usr/src/kernel. Use utility **grep** and the Minix documentation to locate the file. The function would have a logical keyword **pick** the process - or **proc** - to run next. The function selects which process to run from a set of queues that hold ready processes, with each queue given a priority level. In the scheduler code, you will observe that the scheduler runs jobs in order of priority from highest to lowest from q = 0 to q = NR\_SCHED\_QUEUES
2. Change the function to include a random selection of a lower-level priority job. Note that if you modify the priority queue imprudently, the operating system will cease to function. Here are some hints:
   1. Adding some randomness to the scheduler will change how the next process will be selected.
   2. Do not skip or rerun the highest priority jobs which are in the highest priority queues (q = 0, 1, and 2). So when q = 0, 1, 2, do not make any changes.
   3. When you reassign the value of q you want to reassign it to a lower priority job, but not to the high priority jobs q = 0, 1, 2
   4. Making the probability too high, will make the kernel run too slow and making the probability too low, will not make a noticeable change in the kernel.
3. Use ftp to transfer the scheduler .c file to your local machine. Edit the file and locate function that the kernel uses to select a process to run on the CPU. Note, the current selection is based on a pure priority order, i.e. the scheduler runs jobs in order of priority from highest to lowest from q = 0 to q = NR\_SCHED\_QUEUES.
4. Your goal is to achieve the following:
   1. Upon attempting to select the next job, modify the selection to add a random possibility of choosing from a different level.
   2. The lower the probability you choose for this, the more consistent the current selection mechanism will be. Some experimentation may be required to select a reasonable probability value.
5. Save changes to the file and put it back to the Minix /usr/src/kernel using ftp. Rebuild the Minix system again and write down your observations.

You will be graded based on both your implementation of this modification, and on how well you explain the mechanism. It is therefore important to realize that this is both an exercise in coding, as well as an exercise in familiarizing yourself with, and understanding to the point of being able to explain, an unfamiliar code base.

**Observing a change**

A second challenge in this assignment is to demonstrate how the scheduler has been modified. You are to note an observable change in behavior between the unmodified and modified schedulers. To achieve this, you may need to write a simple test program (e.g., a simple hello world program that identifies which process is running).

**Additional Resources:**

* Minix Wiki: https://en.wikipedia.org/wiki/MINIX
* Minix user guide: https://wiki.minix3.org/doku.php?id=usersguide:start
* Minix installation guide: https://wiki.minix3.org/doku.php?id=usersguide:doinginstallation

**Requirements to complete the lab**

1. Show the TA your running Minix system.
2. Write up a description of your steps, which files you modified, and your observations.