COEN 177: Operating Systems Lab 2

Part 1:

The first part of this lab was very simple. We had to sort through the MINIX source code and alter the startup screen text. This task entailed editing a printf statement and recompiling the MINIX OS using the make and make install commands. Once familiar with the source code, we were able to move onto the next part of the lab.

Part 2:

The second part of this lab entailed analyzing the process scheduling algorithm and altering it. The purpose of this was to see how the kernel handles processes in regards to their priority and how it manages to schedule everything efficiently. Our task was to make it a possibility, in this case 20%, that a lower priority task could receive a new priority. This priority could be higher or lower than the current priority. From reading and comprehending the code, I figured out how the tasks are prioritized. The higher priority tasks, better identified as system or kernel tasks have are prioritized from 0 to 7. The user tasks for running programs within the operating system itself are prioritized from 8-16. These are the priorities that we are trying to change. The way I changed the priorities can be found in the code, but here is a general overview. I initialized 2 extra variables to hold the values of the original priority, and the altered one. There is also a variable used in the randomness selection for when the priority will actually be changed. The next variables initialized are for the random generation, which is based on the lower bits received from the internal time functions. In the Pick\_Proc function, I generate the random time as soon as we enter the for loop for executing items from the queue based on their priority. I also store the initial priority value, just in case the priority does not change. After that, I introduce an if condition that selects tasks that have low priority. These are the tasks with 8-16 priority. I also have a condition that allows the priority to change only 20% of the time. I chose 20% because it introduces a factor of randomness while ensuring that the priorities do not change too often, which would cause the operating system to crash. After the for loop, the task is then executed with the code that already existed in pick\_proc. The scheduling scheme used in MINIX is priority scheduling, which simply executes tasks and allocates resources based on a tasks priority. To test my changes, I wrote multiple programs that require time with the CPU. These programs print out things like process identifiers (PID), as well as how many times they have executed. These programs had for loops that used the CPU time. For easier testing, I altered my original Lab 1 to include many more fork calls, which allowed me to analyze how the randomness of 20% was changing the execution. If I had more time, I would have implemented a testing method which would record the CPU bound as well as the read and write bound. This would give me an accurate report of how long it took each process to run in terms of system time and real user time.