Lab 4 Answers

Handling the Null Process

In my implementation, I handle the null process by checking the pid when I dequeue a process from the multilevel feedback queue. If the pid is 0, indicating the null user process, I enqueue it again and call a second dequeue. This allows processes to still be priority 0, and if the null process comes up instead of a valid process then it is skipped. Dequeuing and enqueuing again will still allow the null process to run, but only if it is the only item in the list. Otherwise, it is skipped.

CPU Bound Only

Fair scheduling was implemented properly. This is demonstrated by the fact that as a process executed, its priority decreased, and its time slice increased. Once the process's priority dove below the xts_checkprio() returned value the process was context switched out and a new once began. Once the processes all hit 0 priority, there was a consistent pattern and time slice for each process that demonstrates a proper round robin.

IO Bound Only

Fair scheduling was implemented properly. This was demonstrated by the fact that the priority of each process rose, and the time slice fell as they were switched out. The order the processes were executing in stayed the same once the priorities all rose to 59 so the round robin concept was implemented properly. Also, the null process ran when all processes were not using the CPU.

CPU and IO Bound

Fair scheduling was implemented properly. This is demonstrated by the fact that the integrities confirmed by the CPU bound and IO bound processes were still maintained even while mixing the two. It is also demonstrated by the fact that after all the processes relinquished the CPU, the would execute as soon as the set number of milliseconds passed (after the CPU process finished its time slice). This means that response time for user inputs are responsive and are taking a higher priority than the CPU bound processes.