Mark Yang

Lab 3: Time Share (TS) Scheduling in a Modern Operating System

Modifications to XINU (from what I remember):

Every time resched() is called (wait.c ready.c clkhandler.c kill.c yield.c recvtime.c suspend.c sleep.c receive.c), I would call cpuio() to set the process' priority

I also changed how processes were enqueued in ready.c and used my multilevel feedback queue

Added a prclassify property to the process. Originally was going to use that to determine if process is CPU or IO intensive

Problem 3

3.2)

For newly created processes, I set their initial priorities in create() to 29. That way, processes will start out with the equal priority and eventually reach their modified priority depending on the Solar UNIX dispatch table which was determined by whether or not a process is CPU or IO intensive (if preempt was exhausted or not). For the null process, every time I encountered the null process, I would set its priority to 0 as to keep it from running when other processes are still in the queue.

3.3)

To achieve O(1) time complexity insert and O(c) time complexity dequeue, I used a 2 dimensional array of ROWS by NPROCS. For each element in the 2D array, I initialized all values to -1 to denote that they are empty. I also had another array to store the start and end positions of each queue/row. I then kept updating the start and end everything I enqueued or dequeued that way, to insert, I just insert the process ID into the next available end position of the specific queue/row which takes O(1) time. For dequeue I needed to find which row to dequeue from (starting at the highest priority) which takes O(c) time and just popping off the process ID from the start position.

Problem 4

CPU Intensive output (part of output)

```
CPU PID: 3 Outer Loop: 0 Priority: 0 Remaining Time Slice: 14
CPU PID: 4 Outer Loop: 0 Priority: 0 Remaining Time Slice: 14
CPU PID: 5 Outer Loop: 0 Priority: 0 Remaining Time Slice: 14
CPU PID: 6 Outer Loop: 0 Priority: 0 Remaining Time Slice: 14
CPU PID: 3 Outer Loop: 1 Priority: 0 Remaining Time Slice: 136
CPU PID: 4 Outer Loop: 1 Priority: 0 Remaining Time Slice: 136
CPU PID: 5 Outer Loop: 1 Priority: 0 Remaining Time Slice: 136
CPU PID: 6 Outer Loop: 1 Priority: 0 Remaining Time Slice: 136
CPU PID: 3 Outer Loop: 2 Priority: 0 Remaining Time Slice: 57
CPU PID: 4 Outer Loop: 2 Priority: 0 Remaining Time Slice: 57
CPU PID: 5 Outer Loop: 2 Priority: 0 Remaining Time Slice: 57
CPU PID: 6 Outer Loop: 2 Priority: 0 Remaining Time Slice: 57
CPU PID: 3 Outer Loop: 3 Priority: 0 Remaining Time Slice: 179
CPU PID: 4 Outer Loop: 3 Priority: 0 Remaining Time Slice: 179
CPU PID: 5 Outer Loop: 3 Priority: 0 Remaining Time Slice: 179
CPU PID: 6 Outer Loop: 3 Priority: 0 Remaining Time Slice: 179
CPU PID: 3 Outer Loop: 4 Priority: 0 Remaining Time Slice: 100
CPU PID: 4 Outer Loop: 4 Priority: 0 Remaining Time Slice: 100
CPU PID: 5 Outer Loop: 4 Priority: 0 Remaining Time Slice: 100
CPU PID: 6 Outer Loop: 4 Priority: 0 Remaining Time Slice: 100
```

The multilevel feedback queue dynamically changes the time slices given to each process in according to their priority and preempt used. Therefore, they all round robin and after running a while, the total allocated time to each process is roughly the same but definitely more fair than fair scheduling.

IO Intensive output (part of output)

```
IO PID: 5 Outer Loop: 4 Priority: 58 Remaining Time Slice: 40
IO PID: 6 Outer Loop: 4 Priority: 58 Remaining Time Slice: 40
IO PID: 3 Outer Loop: 5 Priority: 58 Remaining Time Slice: 40
IO PID: 4 Outer Loop: 5 Priority: 58 Remaining Time Slice: 40
IO PID: 5 Outer Loop: 5 Priority: 58 Remaining Time Slice: 40
IO PID: 6 Outer Loop: 5 Priority: 58 Remaining Time Slice: 40
IO PID: 3 Outer Loop: 6 Priority: 58 Remaining Time Slice: 40
IO PID: 4 Outer Loop: 6 Priority: 58 Remaining Time Slice: 40
IO PID: 5 Outer Loop: 6 Priority: 58 Remaining Time Slice: 40
IO PID: 6 Outer Loop: 6 Priority: 58 Remaining Time Slice: 40
IO PID: 4 Outer Loop: 7 Priority: 58 Remaining Time Slice: 40
IO PID: 3 Outer Loop: 7 Priority: 58 Remaining Time Slice: 40
IO PID: 6 Outer Loop: 7 Priority: 58 Remaining Time Slice: 40
IO PID: 5 Outer Loop: 7 Priority: 58 Remaining Time Slice: 40
IO PID: 4 Outer Loop: 8 Priority: 58 Remaining Time Slice: 40
IO PID: 3 Outer Loop: 8 Priority: 58 Remaining Time Slice: 40
```

Just like from the CPU intensive tests, the same goes for the IO intensive processes. They round robin and the priorities and time slices change according to their priority (not so much with IO though since they call sleep and voluntarily give up CPU) and preempt used so the total allocated time to each process is roughly the same (more fair than fair scheduling).

Half and Half (part of output)

```
CPU PID: 3 Outer Loop: 0 Priority: 0 Remaining Time Slice: 194
CPU PID: 4 Outer Loop: 0 Priority: 0 Remaining Time Slice: 194
CPU PID: 5 Outer Loop: 0 Priority: 0 Remaining Time Slice: 195
CPU PID: 6 Outer Loop: 0 Priority: 0 Remaining Time Slice: 190
CPU PID: 3 Outer Loop: 1 Priority: 0 Remaining Time Slice: 194
CPU PID: 4 Outer Loop: 1 Priority: 0 Remaining Time Slice: 194
CPU PID: 5 Outer Loop: 1 Priority: 0 Remaining Time Slice: 195
CPU PID: 6 Outer Loop: 1 Priority: 0 Remaining Time Slice: 199
CPU PID: 3 Outer Loop: 2 Priority: 0 Remaining Time Slice: 193
CPU PID: 4 Outer Loop: 2 Priority: 0 Remaining Time Slice: 194
CPU PID: 5 Outer Loop: 2 Priority: 0 Remaining Time Slice: 195
CPU PID: 6 Outer Loop: 2 Priority: 0 Remaining Time Slice: 198
IO PID: 7 Outer Loop: 0 Priority: 58 Remaining Time Slice: 40
IO PID: 8 Outer Loop: 0 Priority: 58 Remaining Time Slice: 40
IO PID: 9 Outer Loop: 0 Priority: 58 Remaining Time Slice: 40
IO PID: 10 Outer Loop: 0 Priority: 58 Remaining Time Slice: 40
CPU PID: 3 Outer Loop: 3 Priority: 0 Remaining Time Slice: 192
CPU PID: 4 Outer Loop: 3 Priority: 0 Remaining Time Slice: 194
CPU PID: 5 Outer Loop: 3 Priority: 0 Remaining Time Slice: 195
CPU PID: 6 Outer Loop: 3 Priority: 0 Remaining Time Slice: 197
CPU PID: 3 Outer Loop: 4 Priority: 0 Remaining Time Slice: 192
CPU PID: 4 Outer Loop: 4 Priority: 0 Remaining Time Slice: 194
CPU PID: 5 Outer Loop: 4 Priority: 0 Remaining Time Slice: 195
CPU PID: 6 Outer Loop: 4 Priority: 0 Remaining Time Slice: 195
CPU PID: 3 Outer Loop: 5 Priority: 0 Remaining Time Slice: 192
CPU PID: 4 Outer Loop: 5 Priority: 0 Remaining Time Slice: 194
CPU PID: 5 Outer Loop: 5 Priority: 0 Remaining Time Slice: 195
CPU PID: 6 Outer Loop: 5 Priority: 0 Remaining Time Slice: 194
IO PID: 7 Outer Loop: 1 Priority: 58 Remaining Time Slice: 40
IO PID: 8 Outer Loop: 1 Priority: 58 Remaining Time Slice: 40
IO PID: 9 Outer Loop: 1 Priority: 58 Remaining Time Slice: 40
IO PID: 10 Outer Loop: 1 Priority: 58 Remaining Time Slice: 40
```

During the half and half tests, the CPU intensive processes round robin with a high time slice. The OI intensive processes run with a low time slice (the lowest of 40). It is the same as just running CPU and IO intensive processes separately. The allocated time to each process is roughly the same but CPU and IO times are different because of their priorities and because IOs call sleep, they let CPUs run more. So processing times are fair for CPUs and processing times are fair for IOs. The total allocated times are more fair from the time slice scheduling than the regular fair scheduling.

Bonus)

To measure the scheduling overhead of my implementation of time slice scheduling, I brought in my implementation from lab2 the clktimefine. I calculated the milliseconds that resched would run at the start of the function and at the end of the function. Because the amount of time the function took to run was very small, it means that my implementation of the time slice scheduling in constant time plus O(c) time works because I do not take the time to shift all the processes in a single queue/row like a normal queue but rather keep a start and end pointer (head/tail of queue).