Han Wang wang2786(0028451697)

```
Test3.1: Equal Priority(cpu time might be longer since I used mask in test() to make sure output get prints out in a correct format.)


test1(3):
prcputot:2015;clkmilli:8074,prctxswbeg:8073


test2(4):
prcputot:2017;clkmilli:8076,prctxswbeg:8075


test3(5):
prcputot:2019;clkmilli:8078,prctxswbeg:8077


test4(6):
prcputot:2021;clkmilli:8080,prctxswbeg:8079

End of Test3.1

Test3.2: Different Priority


test4.2(28)(10):
prcputot:2015;clkmilli:12035,prctxswbeg:12035


test3.2(27)(9):
prcputot:2015;clkmilli:14055,prctxswbeg:14055


test2.2(26)(8):
prcputot:2015;clkmilli:16075,prctxswbeg:16075


test1.2(25)(7):
prcputot:2015;clkmilli:18095,prctxswbeg:18095

End of Test3.2
```

Question 3

Test 3.1, all four processes are spawn with the same priority of 20, and we can see that all 4 processes finished almost at the same moment(based on clkmilli value). They spent around 2021 ms each to finish the test.

Test 3.2, on the other hand, all four processes are spawn with different priorities(25,26,27,28), and preforming the exact same task as the one before, but this time, “test4.1” which has the highest priority finished first and so on. (name of the processes when creating process was later renamed to 2.1/2.2/2.3/2.4 for better visual cue)

Conclusion, when processes spawned with the same priority, they will share the cpu(round-robin) at the same time to finished the task(by constantly switching in between),while if processes have different priorities, lower priority processes will have to wait until the higher priority task/process is terminated or paused.

This could evidentially cause high priority process with longer/heavier task end-up using the CPU by itself, causing lower priority process to starve, which is a very bad situation that needs to be addressed.

Question 4

The output is included in file Q4_OUTPUT.txt. Here is the sample result:

```
CPUProcess(7):, outer i: 9, Prio: 29092, Preempt: 19,cputot: 501
CPUProcess Finished(7): prcputot:501, prprio:29092, preempt:18
CPUProcess(4):, outer i: 9, Prio: 29653, Preempt: 20,cputot: 461
CPUProcess(4):, outer i: 9, Prio: 29653, Preempt: 19,cputot: 461
```

```
CPUProcess(6):, outer i: 9, Prio: 29613, Preempt: 1,cputot: 481
CPUProcess Finished(6): prcputot:501, prprio:29112, preempt:20
CPUProcess(3):, outer i: 9, Prio: 29615, Preempt: 20,cputot: 487
```

```
CPUProcess(3):, outer i: 9, Prio: 29108, Preempt: 20,cputot: 507
CPUProcess Finished(3): prcputot:507, prprio:29108, preempt:19
CPUProcess(4):, outer i: 9, Prio: 29172, Preempt: 20,cputot: 501
CPUProcess Finished(4): prcputot:501, prprio:29172, preempt:19
CPUProcess Finished(5): prcputot:501, prprio:29152, preempt:20
Test for Q4_5IO begins
```

For Q4_5CPU, all processes exchange and sharing the cpu in an alternated fashion, once any process has use cpu for a while and generated longer cpu total time(20ms interval), the priority keep changing and preempt keep counting down from QUANTUM(20) to (1), and cpu replace the current process with the next process in the queue that has MAXPRIO – CPUTOTALTIME. Processes finished the task in 500 ms

---

```
IO(8):, outer i: 9, Prio: 32766, Preempt: 20,cputot:502
IOProcess Finished(8): prcputot:502, prprio:32766, preempt:19
IO(9):, outer i: 9, Prio: 32766, Preempt: 20,cputot:501
IOProcess Finished(9): prcputot:501, prprio:32766, preempt:19
IOProcess Finished(11): prcputot:501, prprio:32766, preempt:19
IOProcess Finished(12): prcputot:501, prprio:32766, preempt:19
IOProcess Finished(10): prcputot:501, prprio:32766, preempt:19
```

For Q4_5IO, all processes' priority stayed the same and preempt stay the same(20), so all IO processes switched among themselves and they all finished at the same time. That's because we have a sleepms() call in the IO test, which puts current process to sleep stage. It mimics when I/O devices like mouse or keyboard interrupt the current process, since they have a high priority. IOprocess finished around 500 ms.

```
CPUProcess Finished(4): prcputot:499, prprio:32745, preempt:17
IO(8):, outer i: 6, Prio: 32766, Preempt: 20,cputot:344
IO(7):, outer i: 6, Prio: 32766, Preempt: 20,cputot:342
IO(6):, outer i: 6, Prio: 32766, Preempt: 20,cputot:343
CPUProcess(3):, outer i: 9, Prio: 32739, Preempt: 19,cputot: 493
CPUProcess(3):, outer i: 9, Prio: 32739, Preempt: 18,cputot: 493
CPUProcess(3):, outer i: 9, Prio: 32739, Preempt: 17,cputot: 493
CPUProcess(3):, outer i: 9, Prio: 32739, Preempt: 16,cputot: 493
CPUProcess(3):, outer i: 9, Prio: 32739, Preempt: 15,cputot: 493
```

```
IO(6):, outer i: 6, Prio: 32766, Preempt: 20,cputot:344
IO(7):, outer i: 6, Prio: 32766, Preempt: 20,cputot:343
IO(8):, outer i: 6, Prio: 32766, Preempt: 20,cputot:345
CPUProcess(5):, outer i: 9, Prio: 32745, Preempt: 19,cputot: 501
CPUProcess Finished(5): prcputot:501, prprio:32745, preempt:18
CPUProcess(3):, outer i: 9, Prio: 32739, Preempt: 19,cputot: 498
CPUProcess Finished(3): prcputot:507, prprio:32739, preempt:17
IO(8):, outer i: 9, Prio: 32766, Preempt: 20,cputot:502
IOProcess Finished(8): prcputot:502, prprio:32766, preempt:19
IOProcess Finished(7): prcputot:501, prprio:32766, preempt:19
IOProcess Finished(6): prcputot:502, prprio:32766, preempt:19
```

For Q4_3CPU3IO, because IO Processes usually possessed the higher priority, if they are not sleeping, they get to use the CPU without interrupt, other times, they go the sleep and gave way to CPU process to occupy by CPUProcess, within that process, CPU process keep trying to share the CPU among them using QUANTUM time but since the sleep is too short, they end up switch into the next highest process before 20ms. CPU process all finished around the same time, then IO get to finish after them.(Since IO process sleep and CPU process doesn't)

This demonstrated that the scheduling method we implemented is very fair, no process is hugging the CPU and let other process to starve. This shows us I/O devices like mouse get to use the CPU without following the scheduling setup for processes, since the input and output for I/O devices are more important.

Bonus Problem
Ans: After research on this topic, I think Rate Monotonic Scheduling is most fitful for our XINU environment, since our system is based on a Round-Robin scheduler. The Rate Monotonic Scheduling

it schedule processes in a way that none of the threads will ever exceed their deadline, meaning no matter what the load of the system is, there always a utilization based test that guarantees that the system will always be schedulable.

- If a system with one process is 100%
- Then a system of 3 processes is approximately 69%

The utilization based test make sure that if a process passes the test, it will be schedulable and vise versa. In a sense, a RMS scheduling works a lot like the implementation of Q4, which gives the shortest interval process a highest priority.