Lab 3 answers part 2
Benjamin Denison
Username: bdenison

The debug flag is in the xinu header
Set DEBUG to 0 to hide debug messages
Set DEBUG to 1 to show debug messages

4.3.3
Fair scheduling seems to be working. Control of the CPU passes between between processes and at any given time, they've each had about the same amount of time on the CPU. From my output:

```
Pid: 3, outer loop: 396, cpu Total: 1548
Pid: 3, outer loop: 397, cpu Total: 1552
Pid: 3, outer loop: 398, cpu Total: 1556
Pid: 3, outer loop: 399, cpu Total: 1560
Pid: 5, outer loop: 396, cpu Total: 1547
Pid: 5, outer loop: 397, cpu Total: 1551
Pid: 5, outer loop: 398, cpu Total: 1555
Pid: 5, outer loop: 399, cpu Total: 1559
Pid: 6, outer loop: 396, cpu Total: 1547
Pid: 6, outer loop: 397, cpu Total: 1551
Pid: 6, outer loop: 398, cpu Total: 1555
Pid: 6, outer loop: 399, cpu Total: 1559
```

Where the number after "outer loop" is the amount of times the process has passed through the outer loop and the number after "cpu total" is the total time the process has had the CPU. Even at the end of the program, they're very even.

On the null process only executing when there are no others: I wanted to double check that with a debug statement in the null user's while true loop. Sure enough, it only starts printing when all other processes have finished.

```
Pid: 7, outer loop: 396, cpu Total: 1664
Pid: 7, outer loop: 397, cpu Total: 1668
Pid: 7, outer loop: 398, cpu Total: 1672
Pid: 7, outer loop: 399, cpu Total: 1676
Old Prio: 31087, old cpu total: 1680, old id:cpu bound 4, clock value: 8400
uter loop: 399, cpu Total: 1680
Old Prio: 31082, old cpu total: 1685, old id:cpu bound 0, clock value: 8404
677
Old Prio: 31085, old cpu total: 1682, old id:cpu bound 2, clock value: 8405
```

```
It's me! The null user! Wa ha ha!
It's me! The null user! Wa ha ha!
It's me! The null user! Wa ha ha!
It's me! The null user! Wa ha ha!
```

4.3.4

Debug statements make this one a mess, mainly because the null process is called intermittently as the iobound processes simulate waiting on a device. I had to turn UP1 and UP2 way down on this, but it behaves as I'd expect it to, hiccupping out print statements at regular intervals while each process generally has the same amount of CPU time.

```
Pid: 4, outer loop: 34, cpu Total: 118
Pid: 5, outer loop: 35, cpu Total: 106
Pid: 3, outer loop: 35, cpu Total: 108
Pid: 7, outer loop: 35, cpu Total: 108
Pid: 6, outer loop: 35, cpu Total: 110
Pid: 4, outer loop: 35, cpu Total: 121
Pid: 5, outer loop: 36, cpu Total: 109
Pid: 3, outer loop: 36, cpu Total: 111
Pid: 7, outer loop: 36, cpu Total: 111
```

I'm not sure why the first process builds up more CPU time than the other processes. It might be an overhead thing.

4.3.5

Cpu-bound processes still use the same amount of CPU:

```
Pid: 3, outer loop: 205, cpu Total: 782
Pid: 5, outer loop: 203, cpu Total: 770
Pid: 5, outer loop: 204, cpu Total: 773
Pid: 5, outer loop: 205, cpu Total: 777
Pid: 5, outer loop: 206, cpu Total: 781
Pid: 5, outer loop: 207, cpu Total: 785
Pid: 5, outer loop: 208, cpu Total: 789
Pid: 5, outer loop: 209, cpu Total: 793
Pid: 5, outer loop: 210, cpu Total: 781
Pid: 4, outer loop: 206, cpu Total: 785
Pid: 4, outer loop: 207, cpu Total: 789
Pid: 4, outer loop: 208, cpu Total: 793
Pid: 4, outer loop: 209, cpu Total: 797
Pid: 4, outer loop: 210, cpu Total: 801
```

```
Pid: 3, outer loop: 206, cpu Total: 786
Pid: 3, outer loop: 207, cpu Total: 790
Pid: 3, outer loop: 208, cpu Total: 794
Pid: 3, outer loop: 209, cpu Total: 798
Pid: 3, outer loop: 210, cpu Total: 802
Pid: 3, outer loop: 211, cpu Total: 805
Pid: 3, outer loop: 212, cpu Total: 809
```

And io-bound processes still use the same amount of CPU:

```
Pid: 8, outer loop: 38, cpu Total: 115
Pid: 7, outer loop: 38, cpu Total: 116
Pid: 6, outer loop: 38, cpu Total: 116
Pid: 8, outer loop: 39, cpu Total: 118
Pid: 7, outer loop: 39, cpu Total: 119
Pid: 6, outer loop: 39, cpu Total: 119
```

As expected, the cpu-bound processes finish long before the io-bound processes. Io-bound processes shouldn't block the cpu while waiting for input, and instead allow other processes to run while they're waiting, even if the scheduler is supposed prioritize processes that have used the CPU less. Moving from cpu-bound to io-bound processes isn't sequential, however. From the output:

```
Pid: 3, outer loop: 221, cpu Total: 844
Pid: 3, outer loop: 222, cpu Total: 848
Pid: 3, outer loop: 223, cpu Tot: 832
Pid: 4, outer loop: 219, cpu Total: 836
Pid: 4, outer loop: 220, cpu Total: 839
Pid: 4, outer loop: 221, cpu Total: 843
Pid: 7, outer loop: 4, cpu Total: 13
Pid: 8, outer loop: 4, cpu Total: 13
Pid: 6, outer loop: 4, cpu Total: 13
Pid: 4, outer loop: 222, cpu Total: 848
Pid: 4, outer loop: 223, cpu Total: 852
Pid: 4, outer loop: 224, cpu Total: 856
Pid: 4, outer loop: 225, cpu Total: 860
Pid: 5, outer loop: 224, cpu Total: 851
Pid: 5, outer loop: 225, cpu Total: 854
```

In the middle of the cpu-bound processes running, the io-bound processes wake up and are scheduled because they've used far less CPU time. They just sleep again right after, and so take much longer to terminate.

I will note that some of the output has been edited for the sake of readability. I did not use a mutex to prevent print statements from stepping over each other, so some things have been moved to their correct spots to keep things cohesive.

RT Implementation

You could define a new field in the procent struct of type boolean called prisrt and either wrap around or extend create such that you could set the field prisrt to TRUE or FALSE. MAXPRIO (which for me is currently at the 2^15-1 limit) is set to 2^15-2, and upon creation a process with prisrt set to TRUE has its priority set to 2^15-1 (i.e. MAXPRIO+1). In resched (or wherever the priority of a process is updated) skips the step of updating priority if the process in question has its prisrt set to TRUE. Create will call resched and the RT process will be moved to the top of the ready list. Because of the condition in resched, it will never be moved from the top of the ready list. Because it has a priority of MAXPRIO + 1, there can be no process that can be scheduled alongside it unless it too is an RT process.

Alternatively you could extend or wrap around resume, where it does the same thing, but sets the process priority to MAXPRIO + 1. The condition in resched (or wherever the priority of a process is updated) would be changed to skip updating the priority if the process's priority is MAXPRIO+1.