| **MENG PROJECT, FA20** | **Linux Kernel Modifications** |
|---|---|
| Name: Jonathan Gao | NetID: jg992 |

# Progress this week

This week I spent some time looking into the kernel itself. I dug around in the source code a little bit, and am definitely feeling a little lost with the complexity of the source code. I found a videos online and read through some of the Linux kernel documentation to help digest the code a little bit, but I fear that it might be too complicated to try to understand within the time frame I had planned. Hopefully with some more digging I will be able to figure out what's going on.

Some key learning notes for this week:

1. The Linux scheduler right now uses several scheduling policies separated into several classes. These class consist of:

   - STOP
   - DEADLINE
     SCHED_DEADLINE
   - REALTIME
     SCHED_FIFO
     SCHED_RR
   - CFS
     SCHED_NORMAL
     SCHED_BATCH
     SCHED_RR
   - IDLE

   The STOP and IDLE process classes are for CPU load balancing and CPU idle respectively.

2. Every process holds a runqueue for each of the 3 middle classes, where the schedule() function will go through these queues and pick the next process. This schedule() function is found in the core.c file under kernel/sched/

3. The CFS scheduler is implemented without the notion of specifically defined timeslices. Rather, the timeslices are dynamic, completely dependent on the virtual runtime of each process and are decided using a red-black tree to ensure every process will have a fair timeslice on the CPU. Seems like this may be an ideal CPU scheduler, which begs to question why most other operating systems decide to use MLFQ scheduling as opposed to this CFS scheduler. Seeing how CFS is the scheduler currently implemented, would I be able to even implement a MLFQ scheduler that will fit into the kernel and have it operate? This might be a challenge in itself, so this may be where I reconsider the path my project takes.

## Plan for coming week

Dig into existing MLFQ implementations in Linux, or find another way to incorporate ML techniques into the CFS scheduler. The latter might be a little awkward, as there may not be any reasonable way to include such a mechanism.

Primarily, I want to figure out how this scheduler works. There's about 10,000 lines of code in the fair.c implementation, as well as about 5,000 lines of code for each of sched.h and core.c (oof).

## Problems

Perhaps some, as mentioned above.

## Long term plan: still OK?

Hopefully so!