Andrew Gates

Professor Matthew Tolentino

TCES 420

November 17, 2016

Project #3 – Multi-Threaded Scheduling Fun

Machine tested on: Raspberry Pi 3 with 1.2 GHz quad-core ARM Cortex-A53.

Challenges: The main challenge that I faced was successfully communicating between the 3 different threads. I had to set up many variables to tell when a thread was done, when a thread was still running, when a thread’s queue was empty, and so on. This made it difficult to figure out exactly what each thread needed to know from the other threads to finish. Another issue I ran into was with how I set up my queues. My queues allocate a set amount of space and then shift the pointer to the front and to the rear when items get added or removed. I ran into errors when this would go out of bounds from the amount of space that I allocated because I didn’t realize that with the way I set up my queues I need to allocate more room than the amount of jobs that are going to be created to allow for these pointers to shift.

Mutual Exclusion: The way I created Mutual Exclusion between threads is by always locking the queue whenever an item is added to or removed from the queue, and then unlocking it after that. This way if a job was being created and added into the CPU queue, and if a job was finished in IO and sent back to CPU, and if a job was being removed from the CPU queue to run, each of these jobs being enqueued and dequeued would have to wait until the mutex is available and not being used by some other thread. I also created a mutex when creating a job, before I add it to the CPU queue. This way jobs wouldn’t be created at the same time with the same rand() values, and all jobs would be created at different times in order to make sure that all of the specifications of the job were truly random.

I attempted to use Condition Variables to allow for threads to be “woken up” when a new job is added to their corresponding queue. I ran into trouble with doing this though and threads would get stuck waiting for a signal from another thread that is also stuck waiting for a signal. This became too difficult to implement properly so I changed my code to make the threads spin while waiting for a new job to come in.