CMSC 312 - Project Part 3 Report

By Samuel Graves
V00858988

My GitHub: https://github.com/ScythZed
GitHub Repository link: https://github.com/ScythZed/CMSC_312_Project_3

About Part 2:

This project further iterates upon project parts 1 and 2, by implementing a multi-thread, multi-CPU design. I opted to go use C++11's <thread> and <mutex> libraries to implement said changes and restructured my code foundation around these libraries. I decided to go with a set two CPU, two Thread system where each CPU had two unique Threads. Each CPU can choose it's scheduler type.

Scheduler Class:

This is a new class, but with the old code. The idea behind this class was to implement the scheduling that I had in main.cpp and create a class with the schedulers as methods and any pertinent variables as data members. Pre-existing variables like main memory and virtual memory are also made global and defined in the class. There are only three methods within the class. One of which is a print function for main memory, and the other two are the 'Round Robin' and 'Shortest Job First' schedulers as created and expanded upon in parts 1 & 2.

Thread:

I decided to go with the C++11's <thread> library as when doing research online, it looked like the least complex option. This ended up giving me a bit of a headache as I ended up needing to compile the code with the C++11 version instead of the current version of C++(This should be done automatically for you via the makefile). However, when fixing all of the errors related to the version, it works as intended. The thread library allows me to create an object of the class scheduler, run a scheduler of my choosing, and pass in the process list via reference. The only change made to the process class also regards threading. Thread id was added as a data member to each process and initialized to empty. When a thread interacts with a process, it assigns its unique thread id to the process. Threads will only run a process when empty or with their own id. This is what enables multi-threading across a single CPU.

Mutex:

I also used the <mutex> library as it easily worked with the <thread> library once I understood what I was doing. The scheduler class has a static mutex that acts as the global mutex across all of the threads created. The mutex was then locked whenever needed. Notably, whenever writing to main or virtual memory, whenever in a critical section, and whenever printing out to the terminal. This last one is of note as if I had a different way of showing the scheduling and memory allocation this would not be needed. However, when these weren't included the output was too messed up from multiple threads printing at the same time.

How to run the code:

All of the files are linked and compiled via a makefile. Simply run make in the cmd. This should produce a prog1 file. This executable can be run via ./prog1 <infile name>, where <infile name> is the name of the template file. The file I have used to test and is included is called *“input.txt”*. Then you will be prompted to input two parameters in the command line. First, it will ask you how many processes you would like. Enter any positive integer number of processes you would like to run. Each CPU will receive that number of processes. Then you will be asked which scheduler you would like for each CPU. Enter 0 for round-robin or enter 1 for shortest first. The program will then quickly print out how the scheduler would run in the command line output. Furthermore, you are also now prompted if you would like to see Main memory during run time. Note that you can also change the included input.txt file however you see fit. This includes adding a critical section with a ! to denote the start of a critical section, and another ! to denote the end.

Aftermath:

Overall I'm decently happy with how this part of the project went. After completing both all of the project parts and the class itself, I have learned a lot about what I would've done differently given another go around. However, with the parts that I had and the lack of time to do things as the semester comes to an end I am happy to see how my project has progressed. I believe there still exists some undefined behavior within my code; however, upon testing it numerous times with different parameters, the cases are too far between to properly find and fix these behaviors if they do exist. I very much enjoyed and appreciated your class. It was a bit different and difficult but has been one of the stand-out classes I have taken here. Thank you for your time and your teaching.