Operating Systems Concepts Fall 2016 – Project 3 Summary – Douglas Cole Harrison – dch130230

This project was very interesting in that it recommended the use of inheritance and the management of custom data structures in more detail than previous projects. In addition to adding the complexity of the data structures this project also removed the need to manage multiple threads. I had several difficulties related to insufficient language proficiency and reparative refactoring due to fluctuating approached. First, I broke the project down into classes; with each of the OS instance, Scheduler Abstract Class, Job object, two sorting objects, and six concrete scheduler classes extending the abstract one.

Much of the design time was devoted to setting up the environment framework that would call each scheduling object. The main function took care of basics such as: checking command line arguments and returning error messages, turning the text string of the scheduler type into its enum equivalent, passing in the jobs.txt file using an ifstream handle into the OS instance constructor, executing the OS's scheduler, processing the jobs in the scheduler, and reporting the results before cleanup and exit. The OS class parsed the text file and returned a scheduler object initialized and selected based off the interpreted command line options that it was provided by the main thread. The scheduler thread object returned is based off of the scheduler type requested, but is handled by the return type of the abstract scheduler class using polymorphism to abstract the type of thread that main executed. Once the executeScheduler member function of the OS instance has returned the specific scheduler to be used, main calls the scheduler to process its jobs. Job processing populates an array of spaces and job letters which is finally printed when main calls the reportOutput function. Once this framework was finished the project was ready for the schedulers to be implemented.

Many of the difficulties of the project cropped up during the configuration of the frame work and revolved around type errors and nuance bugs typical of first time development with more obscure language components such as inheritance, polymorphism, function pointers, and struct management. I found it very interesting and instructive to refactor the frame work for several iterations until I felt that each component was minimally coupled and logically segregated from its parent and child layers.

Each scheduler has the same basic framework but due to minor differences and relatively compact code blocks I decided to cut and paste functionality to avoid over coupling of the schedulers to one another through common member functions in the main scheduler class. A few difficulties did arise during the scheduler segment of the project development cycle, mainly related to management of the queues. Specifically, the priority queues compare function management was particularly perplexing even after thorough investigation, eventually the issue was solved with individual comparator objects.

The result of this approach was a slow start with momentum that carried me quickly through the last few scheduling algorithms. I was again reminded of the importance of focusing on the entire problem throughout the development cycle to ensure maximum efficiency of the design structure. It is important to leverage similarities where possible, without over investing time in complex logic needed to deal with differences between uses for one method.

I look forward to applying the knowledge and increased C++ capabilities I have honed during this course in my career starting next year. Thank you so much for the encouragement and instruction this semester.