Eric Parris

Noel Overton

4/15/2014

Operating systems

**Preamble**

We decided that it would be best to divide the work up as evenly as possible. The first thing we did was debate the best possible way to implement the classes and create the jobs and memory. We decided that using constructors that randomized and assigned values upon object creation was the most efficient way to perform the task of creating the jobs. We then decided that it would be more efficient to work on the project by ourselves at different times using Dropbox to coordinate between the two of us.

I then began the task of creating the First Come First Serve First Fit method in order to derive a method by which the other two processes could be turned into methods. After that portion was completed, Noel set out to create a rough set of First Come First Serve using Best-Fit method and Shortest Job First using Best-Fit method.

After Noel’s methods were roughly completed, I attempted to run all Three methods at the same time and ran into a massive scope oriented problem. The second and third sorting algorithms started where the preceding one had ended. This was a very odd circumstance considering the design choices that we had made. After spending hours trying to solve the problem, I thought of a solution but it involved modifying the structure of the three methods, the job class, and using an extra text file.

In my opinion, there was relatively little research needed in order to complete our program. The hardest part was mapping out the algorithims for sorting the jobs by the specified criteria as well as solving the scope issue that came up during the implementation of our program.

**Assumptions**

Noel and I assumed that it would be fairly easy to complete the project and that only three classes would have to be created to implement the program. After many hours devoted to scope issues, we decided that is was better to create separate classes for each algorithm as well as using a method of those classes to perform the simulation. We were surprised to find out how difficult it was to complete the program. It was also far more time consuming than expected.

**Data Structure Choices**

We chose create Memory and Job classes because it was fairly obvious that those must be separate because they represent other objects. We also had to create separate classes for each simulation rather than having them all be methods contained within the CS350SemesterProject class due to scope issues. The same scope issues caused us to have to use a fourth output file that contained all of the jobs created prior to the simulations being ran.

**Algorithm Strategies**

Our algorithm strategy was to first create the job and memory classes and make sure that they were working properly. After this, we chose to complete the first come first serve first fit simulation method because it was the simplest and would help us in creating the other two. After this, we elected to complete the first come first serve best fit simulation method because it was very similar to the first come first serve first fit method, differing only in how jobs were placed into the memory.

**Top-down Design:**

After we received the assignment, we had to decide on classes, the methods that would be used in each class and from what class the main would be called. We broke the program into six classes. The Job, Memory and main classes were developed at first. As we continued developing the methods, the length of the code for each method grew. We discovered that the methods that we had created were starting right after another method on the same set of jobs even though we had thought we had coded the program so that each method would take a copy of the same jobs by creating three different arrays with the same jobs in each array. We try our hardest to code around this issue. We found it more feasible and readable to develop classes centered on the different methods that were used. We based those classes on the names of each scheduling strategy. Thus, the FirstComeFirstServeFirstFit, FirstComeFirstServeBestFit, and ShortestJobFirstBestFit classes were created. Comments were included as we developed the methods. After each method was finished, we tested them individually to see how they performed. We discovered we had a lot of debugging to do so that it could perform just as it was supposed to do. Documentation was created after the program was completed.

**Conclusions**

After many hours of effort we can conclude that job sorting algorithms utilizing a best fit method perform better than first fit algorithms most of the time in a simulation such as the one we performed. This assumes that the time taken to calculate the best fit is not factored into the equation, in which case the first fit algorithm might perform better for smaller jobs. When considering our design choices for code, we determined that it probably could have been done in a more concise manner (the fourth .txt file was just to bypass scope issues.)