Daniel Ladner, Alhajji Nyallay, Wei Wu

CSC 332

**Scheduling Algorithms Project**

This goal of this project is to create a program that will simulate various operating system, scheduling algorithms. By doing this, we aim to compare the different scheduling algorithms and determine their comparative advantages and disadvantages. Determining the relative advantages of the currently competing scheduling algorithms is important, because applying the write scheduling algorithm to the right problem will save resources and meet efficiency requirements. This project looks specifically at: first come first serve, shortest job first, and round robin algorithms.

* Challenges

Small errors can cause the compiler to generate dozens of messages, compiler errors are notoriously cryptic and verbose. Correcting compiler errors can be a long tedious process if one has to separately write code, then go through a list of compile errors one by one. We discovered this while writing code using text editors and terminal commands.

Before we start to program as a group, the program design is a real challenge. Knowing how to think about programming is one thing, but the other is knowing how to combine programs together in a way that makes it easier to modify later. Ideas like commenting your code, encapsulation and data hiding and inheritance don't really mean anything when you haven't felt the pain of not having them. Program design is all about making things easier for your future self in the event that you need to add or modify code in the future. In addition, the similar nature of scheduling algorithms means that there is a large amount of reusable code. Coming up with a design that allows reuse of code is difficult, but it is important because it allows changes/corrections to be made to the program without forcing changes to be made to multiple parts of the source code.

Using Visual Studio hindered our progress when we tried to make a cross platform build command. We received various errors due to UTF-16 encoding and windows specific libraries.

A challenge arose as we developed round robin and shortest job first algorithms. These algorithms can not run jobs one at a time like the way “First Come First Serve” can. They need to know all of the jobs that have arrived in order to decide which one to run. As the program was required to run in an evented way (one job at a time), this posed some difficulty.

* How we overcome those challenges.

Using the Visual Studio IDE we were able to spot compile errors while we were writing the code. This was a much more efficient process than running the compiler on the command line and switching between the command line and the text editor to correct errors. This lead to less time searching for errors and more time writing code.

After some consideration, we designed the program to have classes for each scheduling algorithm that are each derived from a base scheduling class. This allowed us to keep each scheduling algorithm modular, while letting us share redundant scheduling algorithm code.

To facilitate the testing/debugging process, we used visual studio’s debugger to quickly step through the code and follow the data through the program. Following the data through the program allowed us to easily see where unexpected data was coming from and fix the corresponding problem.

To make the cross platform makefile we removed all of the windows specific include statements/files and converted all of the file encodings to utf-8.

We overcame the problem with scheduling algorithms where jobs are not run in order, by adding each job to a ready queue when they came and halting the simulation. We would only run the simulation when a job arrives that is in the future. Then, we can continue the simulation for the jobs in the ready queue, because we can be sure that all of the currently relevant jobs are there. At the end, this forced us to tell the scheduling algorithm object that no more jobs were coming so it could complete the jobs in its ready queue.

* In your opinion, what is the best scheduling algorithm?

In my opinion there is no best scheduling algorithm. Each scheduling algorithm has their own advantages and disadvantages. And it will depend on the problem we are facing in order to choose the best algorithm to solve that specific problem. However, of all the algorithms we looked at, Round Robin is the most versatile. Long processes will not lock up the cpu like in FCFS and long programs will never get starved like in SJF. Further, it seems like Round Robin does not sacrifice much in throughput. Its higher CPU utilization seems to allow it to not fall too far behind SJF and FCFS.

# Limitation of Round Robin Scheduling Algorithm

# Round robin scheduling has no priority system. This means if there is a urgent job that needs to get handled it won’t be completed any faster than the other jobs in the queue. Round Robin also wastes CPU time switching between processes.

* What is the effect of decreasing context switch time to 1 time unit?

Decreasing context switch time to 1 time unit will boost performance greatly. Context switching is overhead, therefore with a lower context switch time the processor will spend less time switching between jobs and more time will be spent on productive computing. The effect is larger with round robin which will have the most context switches.

* What is the effect of increasing it to 10?

Higher context switch time will result in more overhead, which will decrease CPU performance. This effect is larger with the round robin algorithm which will have more context switches.

* What percentage of turnaround time is the process waiting?

In our simulation, with a context switch of 5 time units, we found that the percentage of turnaround time spent waiting was about: 50% for SJF and FCFS and 86% for Round Robin with time quantum of 5. SJF has less waiting time than the other algorithms. The smaller the context switch time the smaller amount of time spent waiting, especially for Round Robin.