Aaron Delahoussaye (C00449804)

Allen Sanders (C00496790)

Michael Vedol (C00459436)

Riley Young (C00445412)

Project 3 Report

Task 1: Single Core (Allen Sanders and Michael Vedol)

Runtime Table:

Five Threads: T1-18, T2-7, T3-25, T4-42, T5-21

|  |  |
| --- | --- |
| Algorithm | Completion Time (in milli-seconds) |
| FCFS | 17, 15, 17: AVG = 16 |
| RR | 27, 24, 23: AVG = 25 |
| NPSJF | 19, 20, 20: AVG = 20 |
| PSJF | 22, 19, 18: AVG = 20 |

The Fastest Algorithm: FCFS

Task 2: Multi-Core (Aaron Delahoussaye and Riley Young)

Task 3: Command Line (Riley Young)

The command line accepts -S (1-4) and -C (number) inputs in any order, outputting an error if an invalid input is inputted. The most difficult part was trying to figure out how to make it accept -S and -C in any order, but other than that it was fairly easy to set up.

Task 4: Report

1. What algorithm was the most difficult to implement for a single-core system and for a multi-core system?

For single-core systems, the most difficult to implement was the Preemptive Shortest Job First Algorithm. For multi-core systems, the most difficult was…

1. In your own words, explain how you implemented each task. Did you encounter any bugs? If so, how did you fix them? If you failed to complete any tasks, list them here and briefly explain why.
2. What sort of data structures and algorithms did you use for each task?

Task 1:

* Queue – used to act as the ready queue for TaskThread objects. Used poll() and add() to remove elements from the front of the queue and adding noncompleted elements to the back of the queue
* Semaphores – used to protect access to the ready queue and access for task to run in the task thread.
* Priority Queue – implemented in PSJF algorithms

Task 2: