Group	: ForkQueue
Course	: CS 149-01
Instructor	: Ahmed Ezzat
Homework	: #2

Processor Scheduling Algorithms Homework #2

June 28th, 2016

Student	**Completed**
Tyler Jones (009645610)	X
Scot Matson (009602502)	X
Francisco McGee (008973445)	X
John Kennedy (001462826	X
Daniel Tam (009238632)	X

Group	: ForkQueue
Course	: CS 149-01
Instructor	: Ahmed Ezzat
Homework	: #2

Homework #2 requires creation of code to implement different process scheduling algorithms, measure their performance against randomly chosen input data, and use the data to determine strengths, weaknesses, and suitabilities for particular process workflows. Below are the results obtained and subsequent conclusions drawn.

The method of testing specified by the homework assignment gave guidelines for the testing process. A summary of those guidelines follows:

1) Generate a set of simulated processes to be scheduled by each algorithm. Each simulated process must have:
    a) An arrival time
    b) An expected run time
    c) A process priority
2) Test the following algorithms
    a) First Come First Served (FCFS)
    b) Shortest Job First (SJF)
    c) Shortest Remaining Time (SRT)
    d) Round robin (RR)
    e) Highest Priority First (HPF)
3) Assume only one process queue
4) A maximum of four priority levels may be used for HPF
5) Calculate the following statistics
    a) Average turnaround time for the process
    b) Average waiting time for the process
    c) Average response time for the process
    d) Throughput for the algorithm

Group	: ForkQueue
Course	: CS 149-01
Instructor	: Ahmed Ezzat
Homework	: #2

The test code was run 5 times for each algorithm, creating a new set of simulated processes each time. The Java code for the project is included in the bundle (.zip) with this report, along with output of the algorithm test runs. Java was chosen for this project due it’s object oriented nature, which lends itself to simulations of real processes and algorithms, as well as the team's familiarity with the language.

The tabularized summary of the runs and the calculated data is contained in Table 1 below.

	Turnaround	**Wait**	**Response**	**Throughput**
FCFS	45.94271	40.207317	40.207317	0.15941638
SJF	8.483307	5.75	5.75	0.3016416
SRT	6.46689	5.062129	5.0838323	0.33207062
RR	1.0446482	31.502737	0.5489796	**0.9861984**
HPF non-preemptive	27.80099	21.373333	21.373333	0.14368077
HPF preemptive	**0.87550074**	**1.1460341**	**0.3983051**	0.23838384

Our examination of this data yields the following conclusions:

1) Highest Priority First preemptive yielded the fastest turnaround, wait and response times, thought throughput is very low. This indicates that the HPF preemptive algorithm is best suited towards systems requiring high interaction with end users, such as desktops. The drawback for this algorithm is setting the actual priorities. This exercise is highly personal, and left mostly up to the end user. OS designers can make some assumptions, such as updating video is more important than calculating numbers, but each of these decision points have tradeoffs which will eventually affect the user experience. A high

degree of familiarity with the end user needs and goals is required to properly implement such an algorithm.

2) Removing preemption from HPF yields undesirable results. Although it might be intuitively clear that allowing preemption is necessary for priority to be effective, the results of the test indicate the high degree to which this change impacts the system. Removing preemption yields the second worst turnaround, third longest wait and response times, and lowest throughput. HPF without preemption is inferior to just about every other algorithm.
3) The highest process throughput and shortest turnaround time was obtained through the use of Round Robin. The Round Robin algorithm works for interactive systems like an HPF preemptive system with a fixed arrival time for processes: each process gets a certain fixed time quanta to run, then is preempted by other needy processes. This is effective not only for interactive systems (though not as effective as HPF preemptive), it also yielded the highest throughput of any of our algorithms, making it highly suitable for batch processing as well. **CAUTION**: These results do not take into account high run time jobs. Although more jobs may be scheduled on the processor than would be in a First Come First Serve system, if the number of processes to be performed is high, and arrival times are less than the quanta, and the run times are greater than the quanta, then the chances of a process being completed decline. In such a scenario, FCFS may well be a better choice for a batch system.

In conclusion, we find the Round Robin algorithm to be the easiest choice to make for an interactive or batch system, due to it’s performance, ease of implementation, and simple management. Highest Priority First preemptive is the best choice for interactive systems, but requires more management and regular adjustment in order to yield optimal gains.