Assignment 3 – CPU Scheduling Analysis

Nick Simmons

T00033019

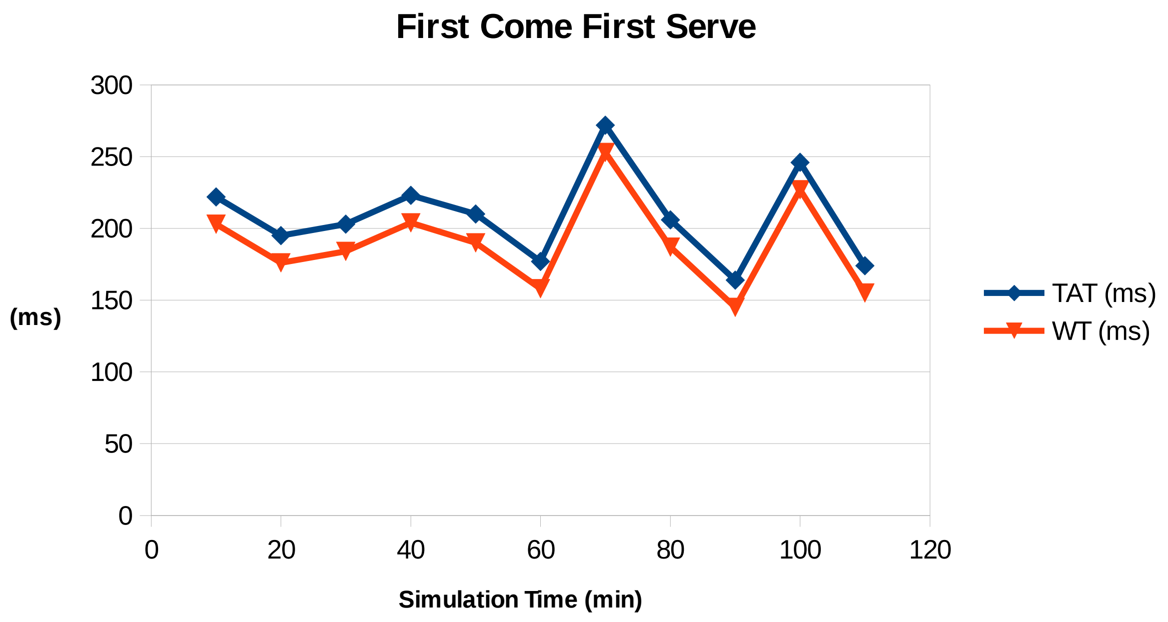
A simulation data set of processes was constructed using the provided SampleGenerator.java with the following configuration:

```
MIN_PRIORITY = 10;
MAX_BURST = 50;
SIM_TIME = 7200000;
AVG_ARRIVAL_TIME = 20;
```

Methodology

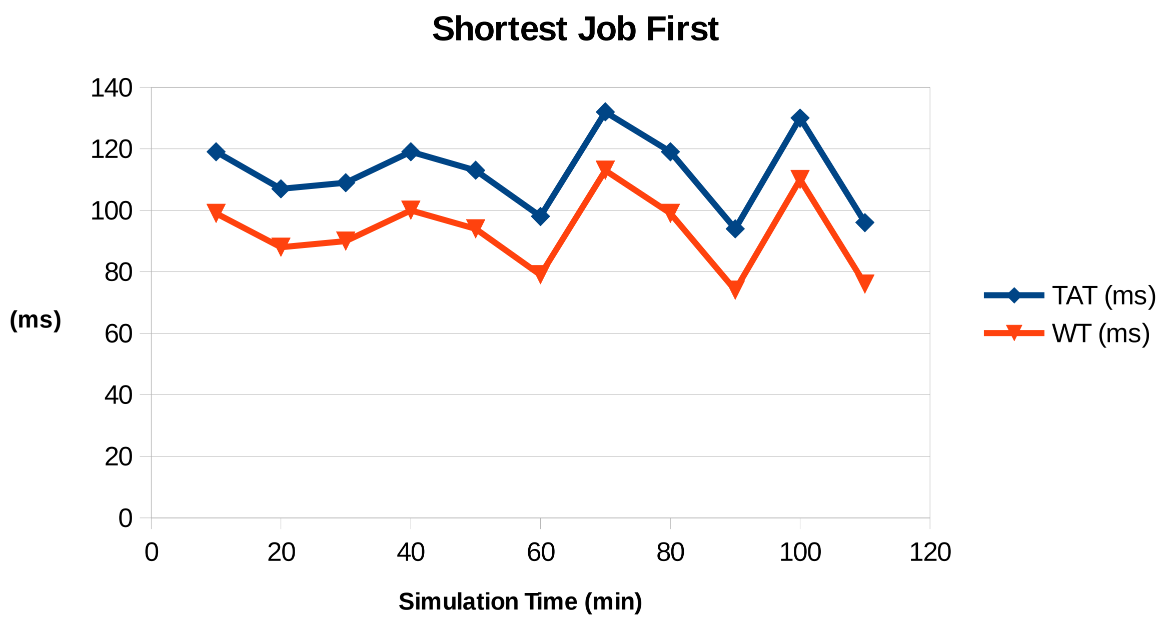
1. First Come First Serve

Processes were input from the simulation file previously generated. A ready queue was constructed prioritized on process arrival time, upon which processes were enqueued at their arrival time; the oldest process at the head of the queue. Processes then were dequeued from the priority queue to the CPU for the predetermined CPU burst time. Once CPU burst time had expired, the CPU process was discarded and the next ready queue process was dequeued to the CPU. This cycle continued until all processes had been loaded, queued and processed by the CPU. Statistics, average turn around time and average waiting time, were computed every 1 minute of simulation time, with a moving average computed every 10 minutes of simulation time.



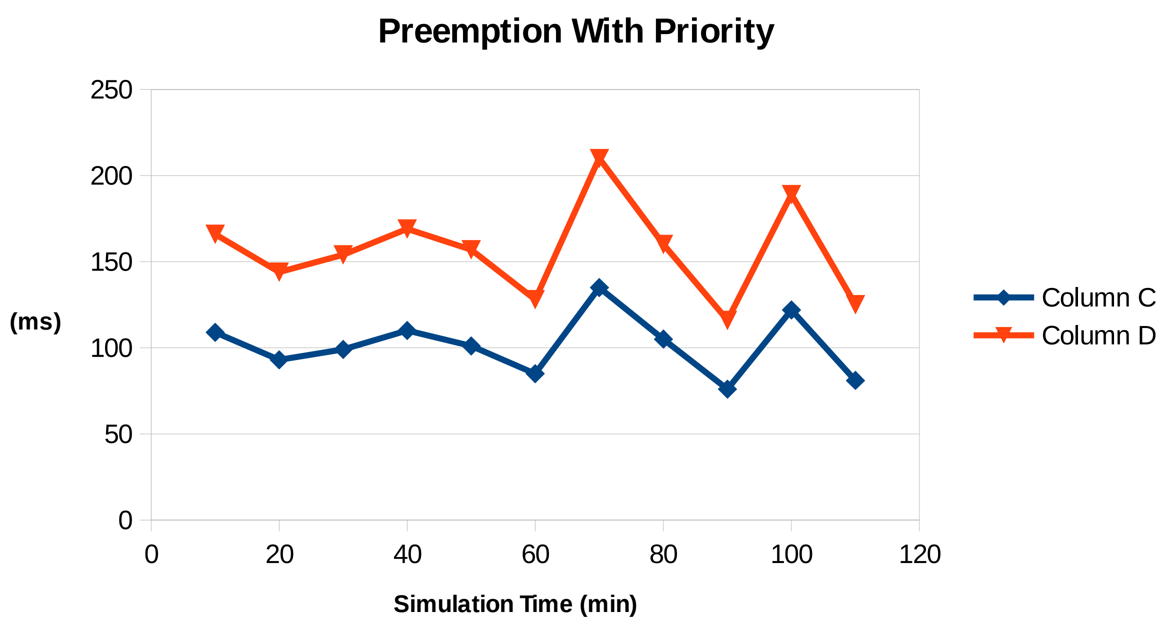
2. Shortest Job First.

Processes were generated previously with a predetermined CPU burst time. A ready queue was constructed prioritized on CPU burst time value; the lowest at the head of the queue. Processes were then dequeued from the priority queue to the CPU for the predetermined CPU burst time. Once CPU burst time had expired, the CPU process was discarded and the next ready queue process was dequeued to the CPU. This cycle continued until all processes had been loaded, queued and processed by the CPU. Statistics, average turn around time and average waiting time, were computed every 1 minute of simulation time, with a moving average computed every 10 minutes of simulation time.



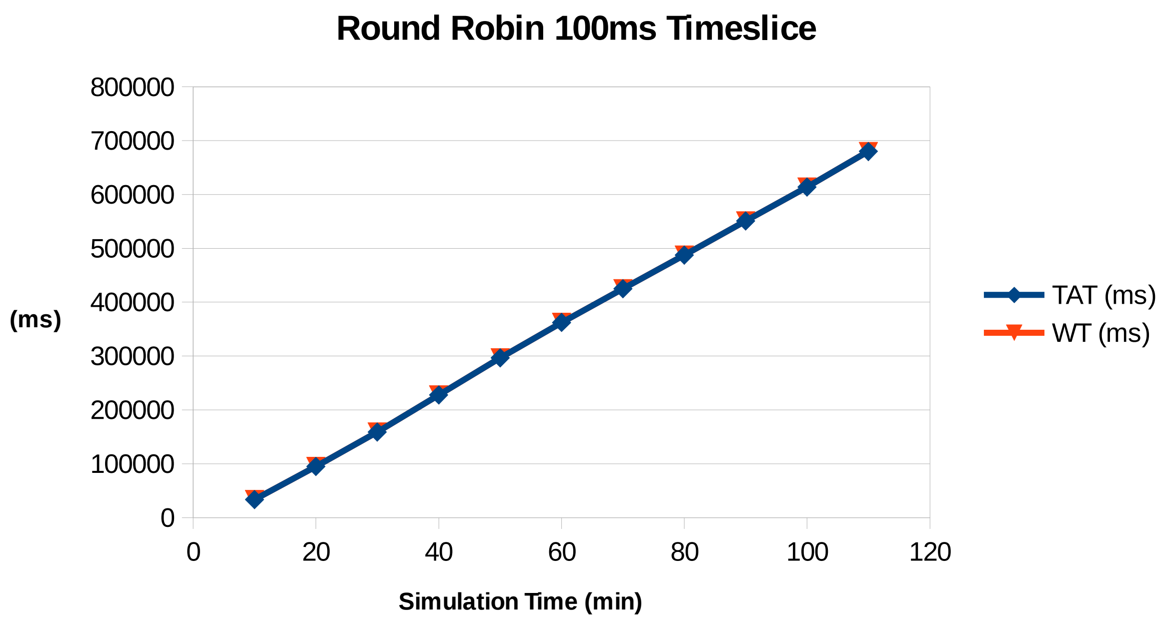
3. Preemption with Priority

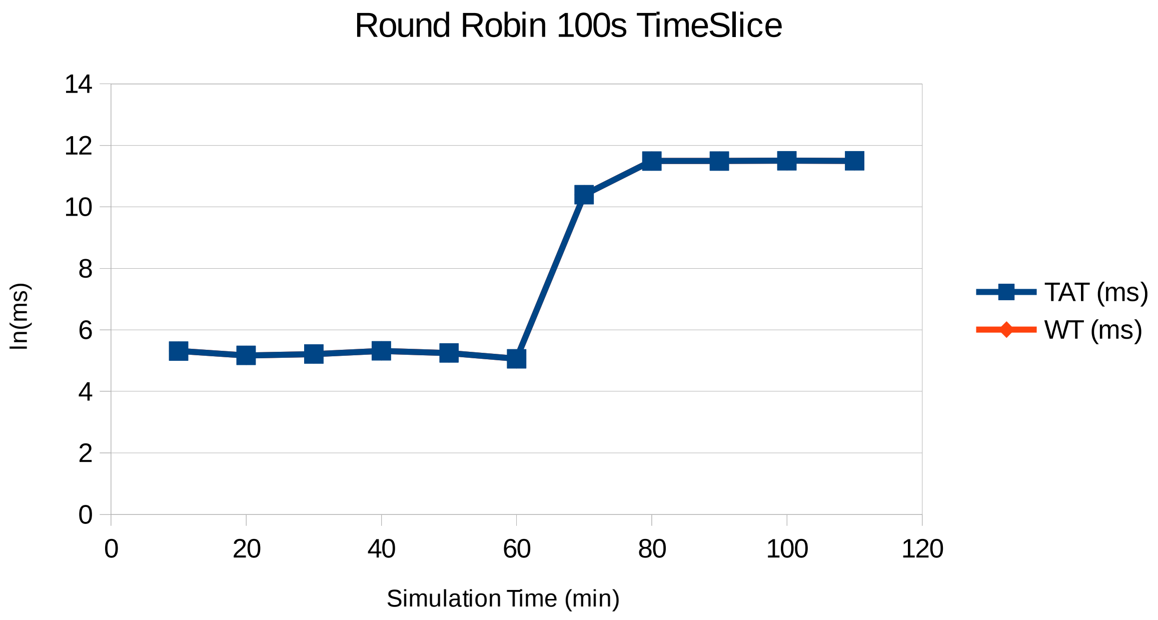
Processes were generated previously with a priority value ranging from 0 or highest priority to 10 or lowest priority. A ready queue was constructed prioritized on priority value; the highest priority process at the head of the queue. Processes were then dequeued from the priority queue to the CPU. Upon the subsequent time index, the priority of the currently running CPU process was compared to the next process on the ready queue. If cpu.getPriority() > readyQ.getHighestPriority() evaluates true a context switch is executed. The remaining CPU burst time of the executing process is recorded, register retention in essence, and the CPU process is returned to the ready queue. The higher priority queued process is then dequeued to the CPU to begin execution. The priority algorithm then repeats. Statistics, average turn around time and average waiting time, were computed every 1 minute of simulation time, with a moving average computed every 10 minutes of simulation time.

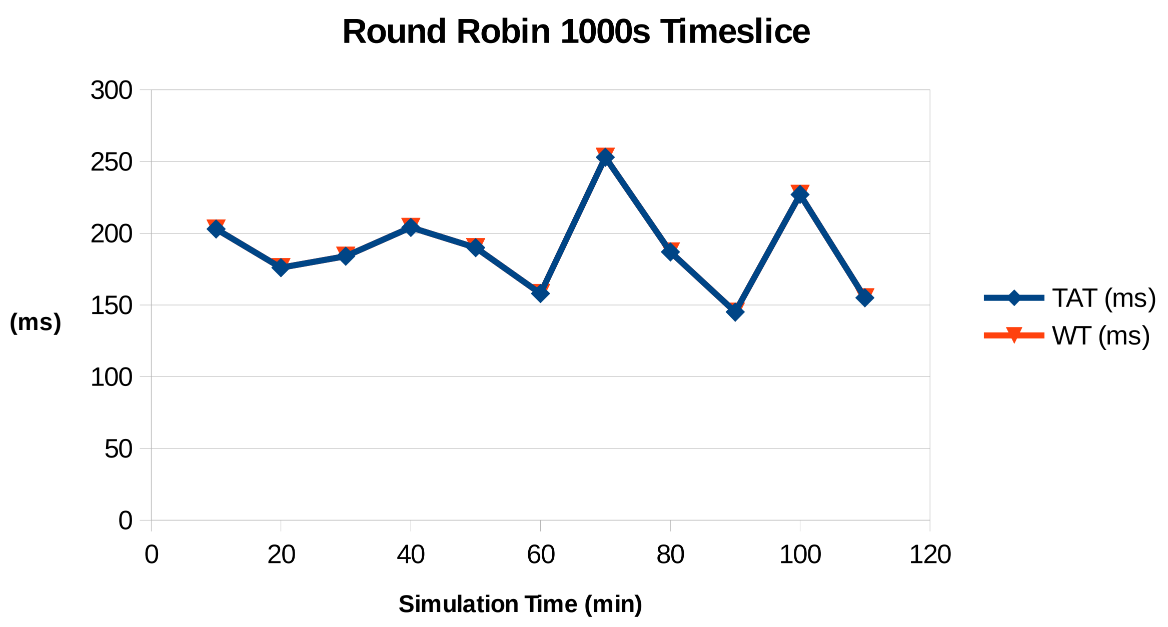


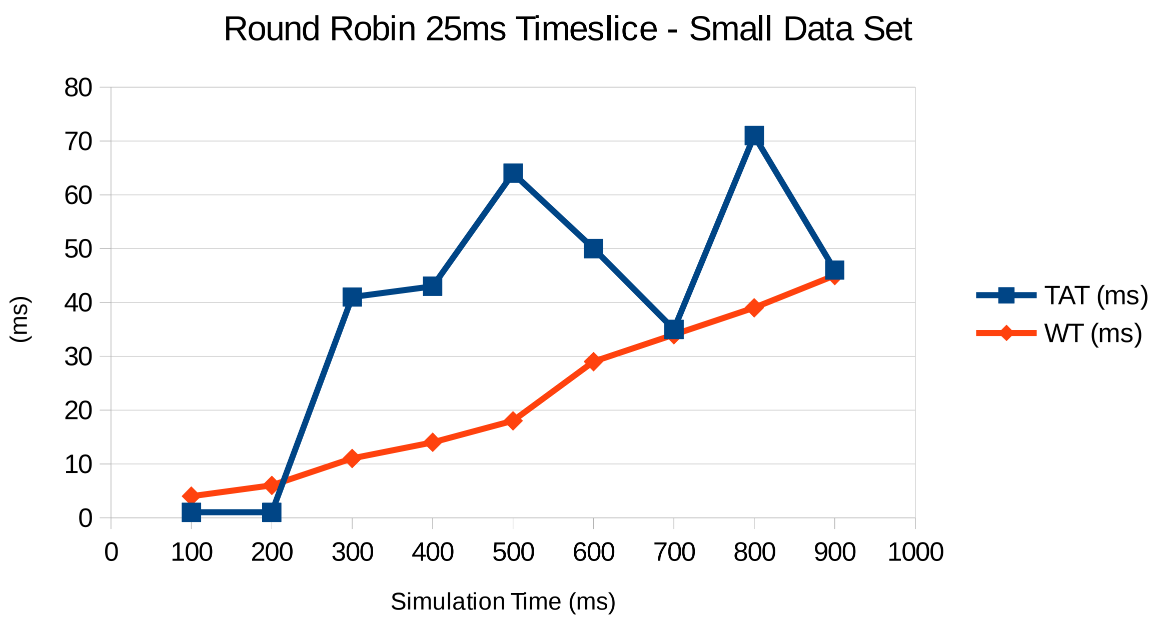
4. Round Robin

Processes were input from the simulation file previously generated. A ready queue was constructed prioritized on process arrival time, upon which processes were enqueued at their arrival time; the oldest process at the head of the queue. Processes then were dequeued from the priority queue to the CPU and are allowed to execute for a predetermined interval or time-slice. After each time-slice interval the CPU executes a context switch, storing the remaining burst time, and returning the process end of the queue. The CPU then fetches the next queued item and executes for an additional time-slice before repeating the algorithm once again. Statistics, average turn around time and average waiting time, were computed every 1 minute of simulation time, with a moving average computed every 10 minutes of simulation time. Due to the dynamic nature of the time-slice interval, multiple simulations were made with distinct results.






Round Robin 1000s Timeslice
300
250
200
150
100
50
0
(ms)
0
20
40
60
80
100
120
Simulation Time (min)
TAT (ms)
WT (ms)


Round Robin 25ms Timeslice - Small Data Set
80
70
60
50
40
30
20
10
0
(ms)
0
100
200
300
400
500
600
700
800
900
1000
Simulation Time (ms)
TAT (ms)
WT (ms)

Analysis

1. First Come First Serve

    A mean turn around time of 208.36ms with a standard deviation of 31.96ms was recorded. A mean waiting time of 189.27ms with a standard deviation of 31.96ms was recorded.

2. Shortest Job First

    A mean turn around time of 112.36ms with a standard deviation of 12.96ms was recorded. A mean waiting time of 92.90ms with a standard deviation of 13.00ms was recorded.

3. Preemption with Priority

    A mean turn around time of 101.45ms with a standard deviation of 17.62ms was recorded. A mean waiting time of 156.18ms with a standard deviation of 27.90ms was recorded.

3. Round Robin 100ms

    A mean turn around time of 357302.63ms with a standard deviation of 215038.91ms was recorded. A mean waiting time of 357321.90ms with a standard deviation of 215044.18ms was recorded.

4. Round Robin 100s

    A mean turn around time of ln(t)=7.97ms with a standard deviation of ln(t)=3.18ms was recorded. A mean waiting time of ln(t)=7.97ms with a standard deviation of ln(t)=3.18ms was recorded.

5. Round Robin 1000s

    A mean turn around time of 189.27ms with a standard deviation of 31.96ms was recorded. A mean waiting time of 189.27ms with a standard deviation of 31.96ms was recorded.

6. Round Robin 25ms – Small Data Set

    A mean turn around time of 39.11ms with a standard deviation of 24.35ms was recorded. A mean waiting time of 22.22ms with a standard deviation of 14.96ms was recorded.

Discussion

FCFS, SJF, PWP share similar curvature indicating a bottle neck in processing at the 70min and 100min time index, most likely caused by frequent high burst processes. Preemption with priority displays the fastest turn around time, though shortest job first displays minimal queue waiting time, with first come first serve being the slowest of the three in both categories of measurement. For this simulation, PWP would maximize throughput for a user and SJF maximize CPU utilization and minimization RAM utilization.

Much more interesting are the results of round robin scheduling. For small time slices, even several orders of magnitude greater than maximum burst time, round robin displays considerable lag. Due to the incredible number of processes being queued, it is postulated than some processes load for execution immediately before a context switch and then are removed and placed at the rear of an ever increasing queue containing many thousands of higher priority processes. These unfortunate processes end up waiting thousands of time units greatly skewing the statistics.

Evidence for this theory is presented by increasing the time slice to 100s as shown is the corresponding graph, the natural logarithm of mean time being presented for clarity. Initially all processes are able to complete in a reasonable time frame, but at time index 65min an inflection point occurs with the mean statistics increasing rapidly indicating excessively re-queued processes. This frequent queuing cascades, further exacerbating the problem. Increasing the time slice further to 1000s should allow enough time for even the most intensive process sequence to complete and as shown by the graph the simulation mimics first come first serve with identical mean waiting times and similar mean turn around times.

Clearly a round robin schedule is not appropriate for the simulation data. A second, 1000ms, simulation set was constructed with 47 processes. As shown in the graph a time slice of 25ms is sufficient for efficient scheduling. Possible solutions allowing for round robin would be a reasonable, ~100 item, queue size limit, though this would decrease secondary storage utilization, or careful selection of a time slice value based analysis of future process requirements, if possible.