Group 1: Lab 1 Report
Instructor: Dr. Saad Biaz
Institute: Auburn University
Paul Chong (phc0004@auburn.edu)
Justin Sawyer (jts0047@auburn.edu)
Date Submitted: June 11, 2018

## Policy Evaluation

To gather our data for each policy, we compiled/ran our code in the TUX environment and let it generate 250 processes before we compared our numbers to the provided 'Behind the Scenes' numbers. Our process management numbers (See Appendix A, Chart 1) were recorded in the required table format and the 'Behind the Scenes' data was recorded as well (See Appendix A, Chart 2) to see a general comparison between the two. Before moving on to analyzing the different CPU scheduling policies, we ensured that our code generated accurate data when compared to the 'Behind the Scenes' figures. Although we were expecting discrepancies of 5% or more, we found that our results were surprisingly *very* close instead as most of our numbers had a discrepancy of 0 to .0001% (See Appendix A, Chart 3). Our largest and lone discrepancy by far came from Shortest Job First's average wait time as 4.66%, but we were expecting this as AWT generally has the highest discrepancy and therefore did not treat it as an abnormality. With all this in mind, we thus concluded with confidence in our code that our data was sufficiently accurate and moved on to analyzing the results of the individual CPU scheduling policies.

## Analysis of FCFS (First Come First Serve)

We refer to Appendix A, Chart 1 for this section. We recall from class that FCFS's greatest weakness is that its turnaround time will increase greatly if it happens to run against large processes that come in front of smaller ones. As seen in our data, it is evident that the processes generated in this lab have exposed this weakness as FCFS has a considerably larger turnaround than the other policies of 12.45s.

As FCFS is non-preemptive, processes will be placed onto the CPU much later than their arrival times on average and will thus draw large response times, which is also shown in the chart as 2.81s. Furthermore, the nature of FCFS will cause the time the CPU is busy to be higher since it does not care about how large or small a process is. It will simply keep the CPU busy with every process and not bother it with context switches.

Although FCFS generally has better throughput than SJF, it fell under SJF and had a mere .899. However, given the environment of this lab, generating (not to be confused with completing) many jobs will cause FCFS's throughput to eventually stall and decrease. It thus has the worst throughput out of all the tested policies.

Finally, its greatest weakness explains the largeness of its 11.56 AWT because if the CPU is going through large processes before the shorter ones, the shorter processes will stay longer in the ready queue and will cause AWT to greatly increase.

## Analysis of SJF (Shortest Job First)

We refer to Appendix A, Chart 1 for this section. As SJF is known to have optimal turnaround times, our first inspection was of course – the turnaround times. As expected, SJF had the best turnaround time, with it being at least twice as good as any Round Robin policy and almost four times as good as FCFS.

Since SJF goes through the shortest jobs first, the arrival times will be closer to the times that the jobs are first placed onto the CPU and will produce better response times than FCFS. This is evident in the chart as SJF has almost half the response time of FCFS. However, FCFS and SJF are both non-preemptive so both their response times seem lackluster to pre-emptive policies such as SRTF, RR, and MLFQ.

As noted in the analysis of FCFS, SJF came out to have the best throughput of our lab, which is a bit unorthodox since FCFS has better throughput on average. However, given the environment of our lab where we are generating 250 processes and not completing them, it may be intuitive to understand that generating the shortest jobs first in a 250-process constriction will ultimately allow SJF to come out with the best throughput. This, however, may not be the case if say, the processes generated were to be doubled or tripled.

Due to how SJF takes shorter processes first, most processes will not have to spend a lot of time waiting to be picked up after arrival. This explains the stellar average wait time of SJF when compared to other policies.

## Analysis of RR (Round Robin) and Increasing Quanta

We will be referencing Appendix B for this section. Running RR with different quanta greatly increased our ability to see how the different measurements grew over time. As we can see with our five charts where the measurements were each plotted with the quanta as the functions, growth or decay are never sporadic and are within being slightly linear to almost perfectly linear.

Starting with average turnaround (Chart 1), we can see that turnaround becomes significantly better as the quantum is changed from 1ms to 5ms but only improves marginally while changing from 5ms to 20ms. The big, initial jump can be explained because with so many context switches and bouncing around from one process to the next after a mere 1ms, the turnaround will be negatively impacted as it takes much longer to get to process completion. As the quanta grow, turnaround gets better until about 20ms. Interestingly, we can observe that after 20ms, turnaround time begins to get worse again because too big of a quantum is not a good implementation either.

Chart 2 gives us a clear understanding of how RR's response times are affected by the quanta because it is almost completely linear. As the quantum is increased, response times worsen as well because a big quantum means that a process will have to wait a long time for its turn on the CPU. With small quanta, the processes do not have to wait long and response times are thus lowered.

Since throughput can be directly correlated with the amount of context switches, it is no surprise to see that a small quantum of 1, which has tremendous numbers of context switches, causes throughput to suffer greatly when compared to bigger quanta (Chart 3). However, it is important to note that for our lab and environment, throughput does not grow linearly and begins

to level out after a quantum of 10. This may be explained by how big or small the processes to be generated may have been. If the processes were exponentially bigger than 50, we may have seen a different chart.

Unlike throughput, we see an almost inverse relationship with wait time as the quantum is increased. If the quantum is too small, the CPU will be busy doing small context switches and will not be able to reach other processes waiting in the ready queue in a timely manner. Increasing the quantum will thus allow the CPU to take bigger chunks out of individual processes and get to the ones that are waiting. However, as seen in Chart 4, if the quantum gets too big, wait times may decrease instead because it will then turn to a situation where the CPU is too busy taking large chunks out of certain processes and not get to the rest in time.

Lastly, as the quanta grow, we can simply state that the CPU Busy Time will grow as well because the CPU will be 'busier' while taking bigger chunks out of the given processes. With smaller quantum, the CPU will not be as busy because it will be taking more time doing context switches than eating at the processes themselves.

## Conclusion

With all these observations, we can close our thoughts with the most notable features that have been produced by this lab. Our code can be considered accurate because, besides our expected and high 4.66% discrepancy in our SJF's average wait time, our process management numbers consistently stayed within discrepancies of .0001% to 1%.

Furthermore, the concepts that we have learned in class align with many of our results, such as SJF having an optimal turnaround time or FCFS having longer response times compared to other policies. The exceptions that have arisen, such as SJF having a higher throughput than FCFS, can be logically explained by discussing the environment that we are operating in.

We were also able to establish clear relationships between growing quanta and the various measurements of turnaround time, response time, CPU busy time, throughput, and waiting time. As we put in bigger quantum numbers, we were satisfied to see that turnaround and wait time decreased as desired while response time, throughput, and CPU busy time grew.

This lab allowed us to observe both non-preemptive and preemptive CPU scheduling policies and how they differently handle processes. Most importantly, it taught us more about how these policies came to be and why it was and still is vital for more complex methods to be created for the sake of efficiency and convenience.

# Appendix A

**Chart 1**

*Process Management Numbers*

| Policy | Policy Number | TAT | RT | CBT | T | AWT |
|---|---|---|---|---|---|---|
| FCFS | 1 | 12.454190 | 2.813439 | 0.901465 | 0.899256 | 11.563672 |
| SJF | 2 | 3.560274 | 1.551910 | 0.879360 | 1.039176 | 2.732518 |
| RR (Q = 1ms) | 3, 1 | 9.466193 | 0.015274 | 0.845485 | 0.914483 | 8.688267 |
| RR (Q = 5ms) | 3, 5 | 7.441728 | 0.046451 | 0.891524 | 0.961968 | 6.613411 |
| RR (Q = 10ms) | 3, 10 | 7.180118 | 0.085942 | 0.898238 | 0.970565 | 6.351062 |
| RR (Q = 15ms) | 3, 15 | 7.091887 | 0.125775 | 0.900396 | 0.970543 | 6.262837 |
| RR (Q = 20ms) | 3, 20 | 7.037191 | 0.164637 | 0.901993 | 0.970521 | 6.208157 |
| RR (Q = 25ms) | 3, 25 | 7.056973 | 0.202729 | 0.902401 | 0.970543 | 6.227966 |
| RR (Q = 50ms) | 3, 50 | 7.160140 | 0.393217 | 0.904147 | 0.970564 | 6.331167 |

**Chart 2**

*Behind the Scenes Numbers*

| Policy | Policy Number | TAT | RT | CBT | T | AWT |
|---|---|---|---|---|---|---|
| FCFS | 1 | 12.454191 | 2.813439 | 0.905429 | 0.899261 | 11.562369 |
| SJF | 2 | 3.560275 | 1.551911 | 0.882868 | 1.039177 | 2.604971 |
| RR (Q = 1ms) | 3, 1 | 9.466193 | 0.015274 | 0.845507 | 0.914501 | 8.567071 |
| RR (Q = 5ms) | 3, 5 | 7.441728 | 0.046452 | 0.891559 | 0.961982 | 6.585154 |
| RR (Q = 10ms) | 3, 10 | 7.180118 | 0.085942 | 0.898292 | 0.970576 | 6.336232 |
| RR (Q = 15ms) | 3, 15 | 7.091887 | 0.125775 | 0.900471 | 0.970554 | 6.252482 |
| RR (Q = 20ms) | 3, 20 | 7.037191 | 0.164637 | 0.902087 | 0.970529 | 6.200681 |
| RR (Q = 25ms) | 3, 25 | 7.056973 | 0.202730 | 0.902517 | 0.970552 | 6.221700 |
| RR (Q = 50ms) | 3, 50 | 7.160140 | 0.393217 | 0.904368 | 0.970570 | 6.328455 |

**Chart 3**

*Discrepancy Between Behind the Scenes and Process Management Numbers (Displayed as percentage)*

This was calculated with the following formula. Significant units were not modified:

((Behind the Scenes Cells – Process Management Cells) / Process Management Cells) * 100

| Policy | Policy Number | TAT | RT | CBT | T | AWT |
|---|---|---|---|---|---|---|
| FCFS | 1 | 8.029E-06 | 0 | 0.439729 | 0.000556 | 0.011268 |
| SRTF | 2 | 2.808E-05 | 6.44E-05 | 0.39892 | 9.623E-05 | 4.667746 |
| RR (Q = 1ms) | 3, 1 | 0 | 0 | 0.002602 | 0.001968 | 1.394939 |
| RR (Q = 5ms) | 3, 5 | 0 | 0.002153 | 0.003926 | 0.001455 | 0.427268 |
| RR (Q = 10ms) | 3, 10 | 0 | 0 | 0.006012 | 0.001133 | 0.233504 |
| RR (Q = 15ms) | 3, 15 | 0 | 0 | 0.00833 | 0.001133 | 0.16534 |
| RR (Q = 20ms) | 3, 20 | 0 | 0 | 0.010421 | 0.000824 | 0.120422 |
| RR (Q = 25ms) | 3, 25 | 0 | 0.000493 | 0.012855 | 0.000927 | 0.100611 |
| RR (Q = 50ms) | 3, 50 | 0 | 0 | 0.024443 | 0.000618 | 0.042836 |

# Appendix B

**Chart 1**

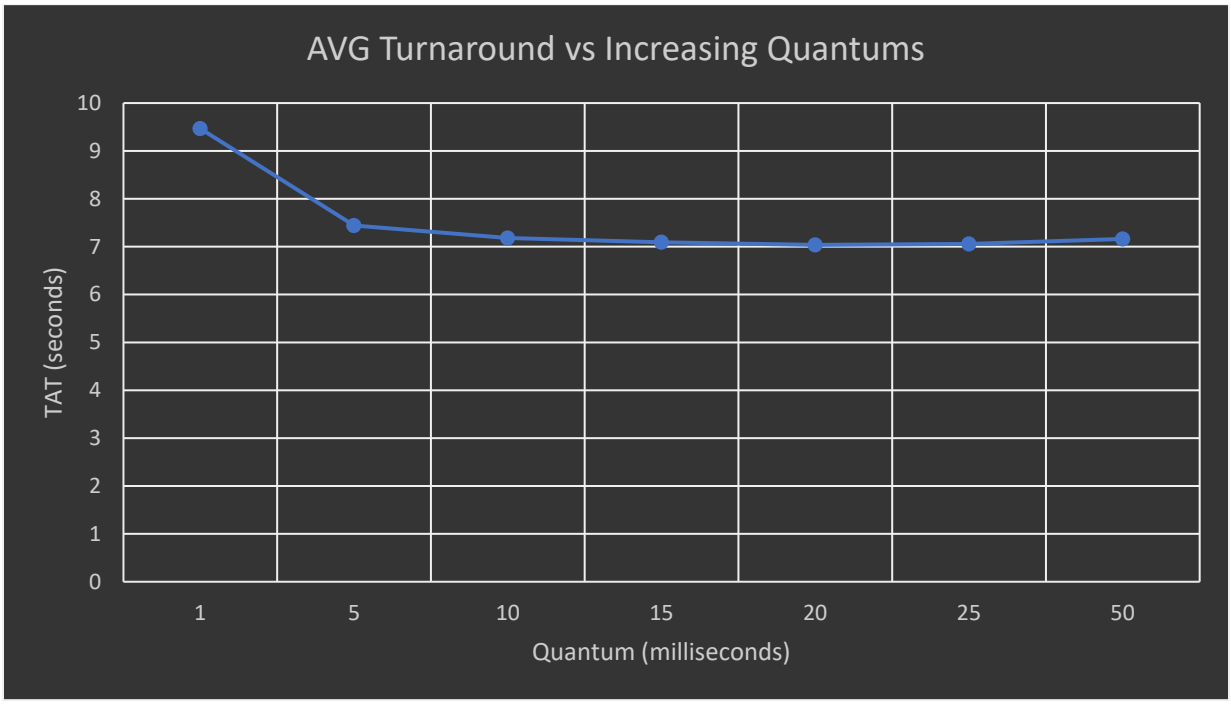



**Chart 2**

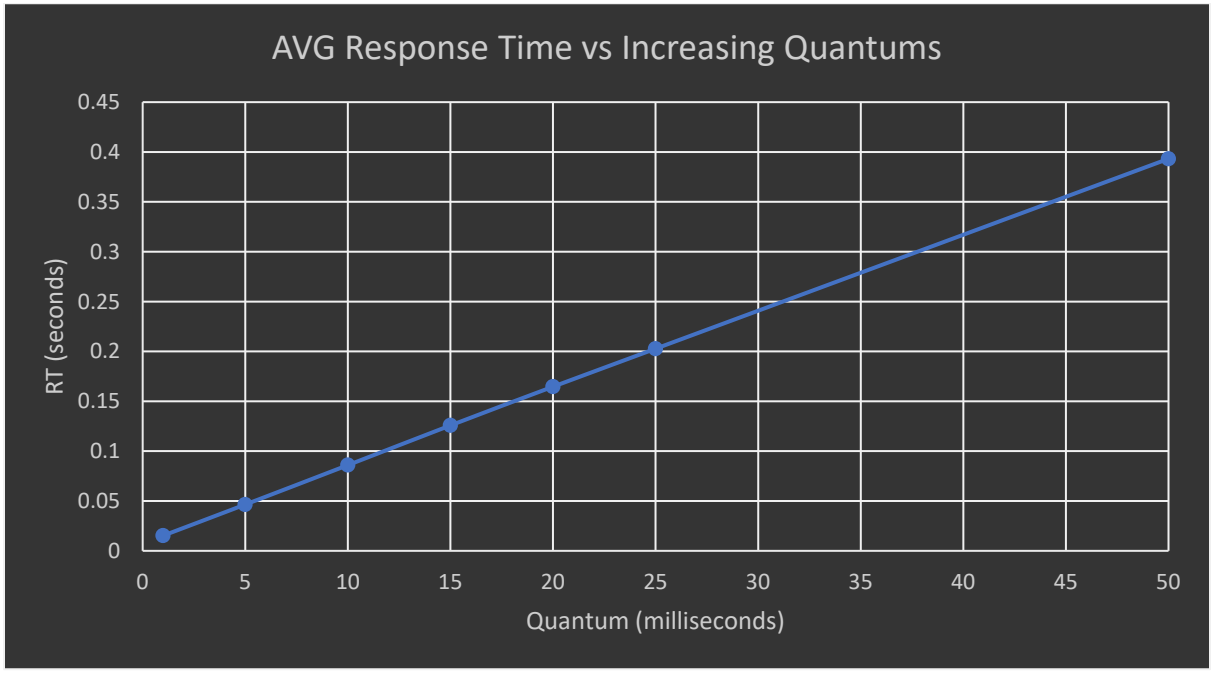


**Chart 3**

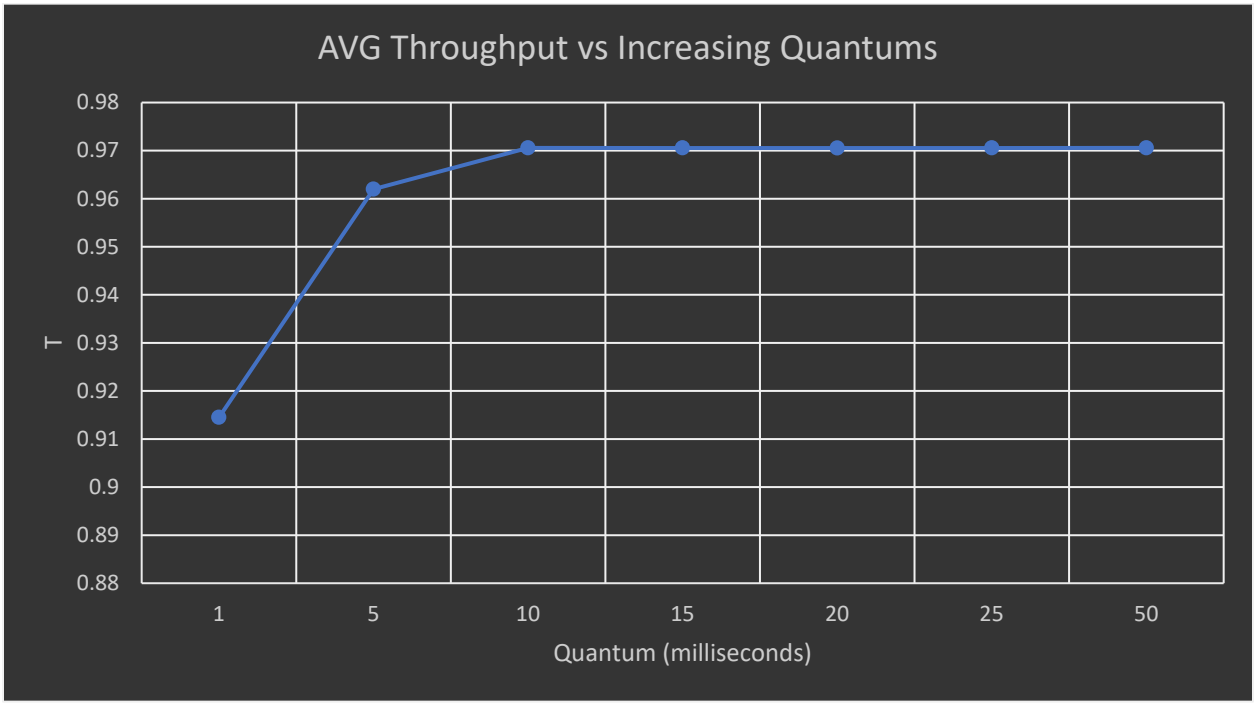



**Chart 4**

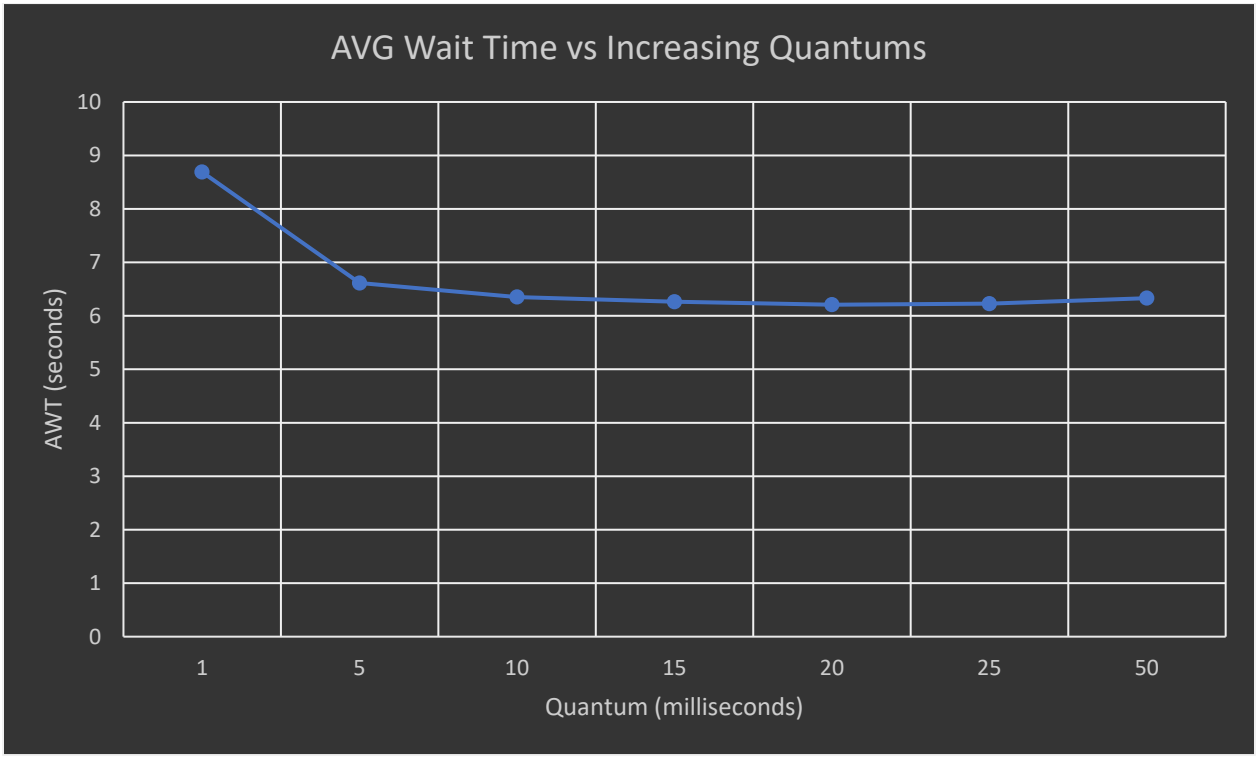
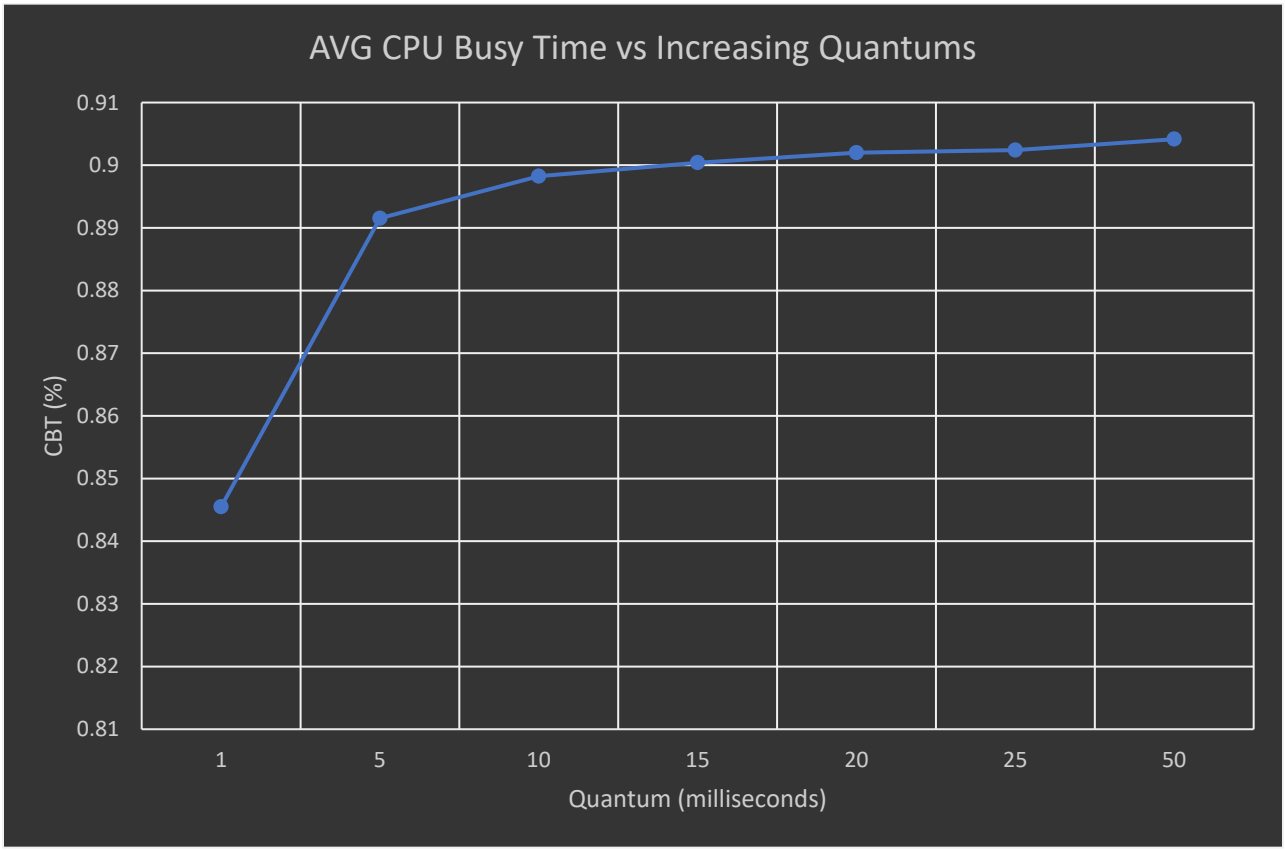


**Chart 5**


AVG CPU Busy Time vs Increasing Quantums
0.91
0.9
0.89
0.88
0.87
0.86
0.85
0.84
0.83
0.82
0.81
CBT (%)
1
5
10
15
20
25
50
Quantum (milliseconds)