# Linux Schedulers

# Programming Assignment Four

# Jacob Berman

# November 14th 2016

# CSCI 3753 - Operating Systems - Shivakant Mishra

**Abstract:**

This report contains an investigation of the schedulers SCHED\_OTHER, SCHED\_FIFO, and SCHED\_RR in a Linux environment.  The schedulers were tested with different conditions such as if the process type, load of instances, and priority levels. Through the investigation it was found that no one scheduler is best suited across the board however certain schedulers do scale better than others.

**Introduction:**

There are a few types of Linux schedulers and my investigation is to figure out which schedulers work in the best conditions. To begin my investigation, I wrote a test program taking in arguments to test these conditions. The conditions my program took in were the scheduler type, load of processes, and if the priorities for these processes would be uniform or not. I then proceeded to run these conditions with programs that were either CPU intensive, IO intensive or mixed. I created benchmarks such as wall times, user mode CPU in seconds, kernels mode CPU in seconds, CPU percentage, voluntary context switches, and non-voluntary context switches. Using the data, I collect using these benchmarks I will be able to draw some conclusion on Linux schedulers and how they act in certain conditions.

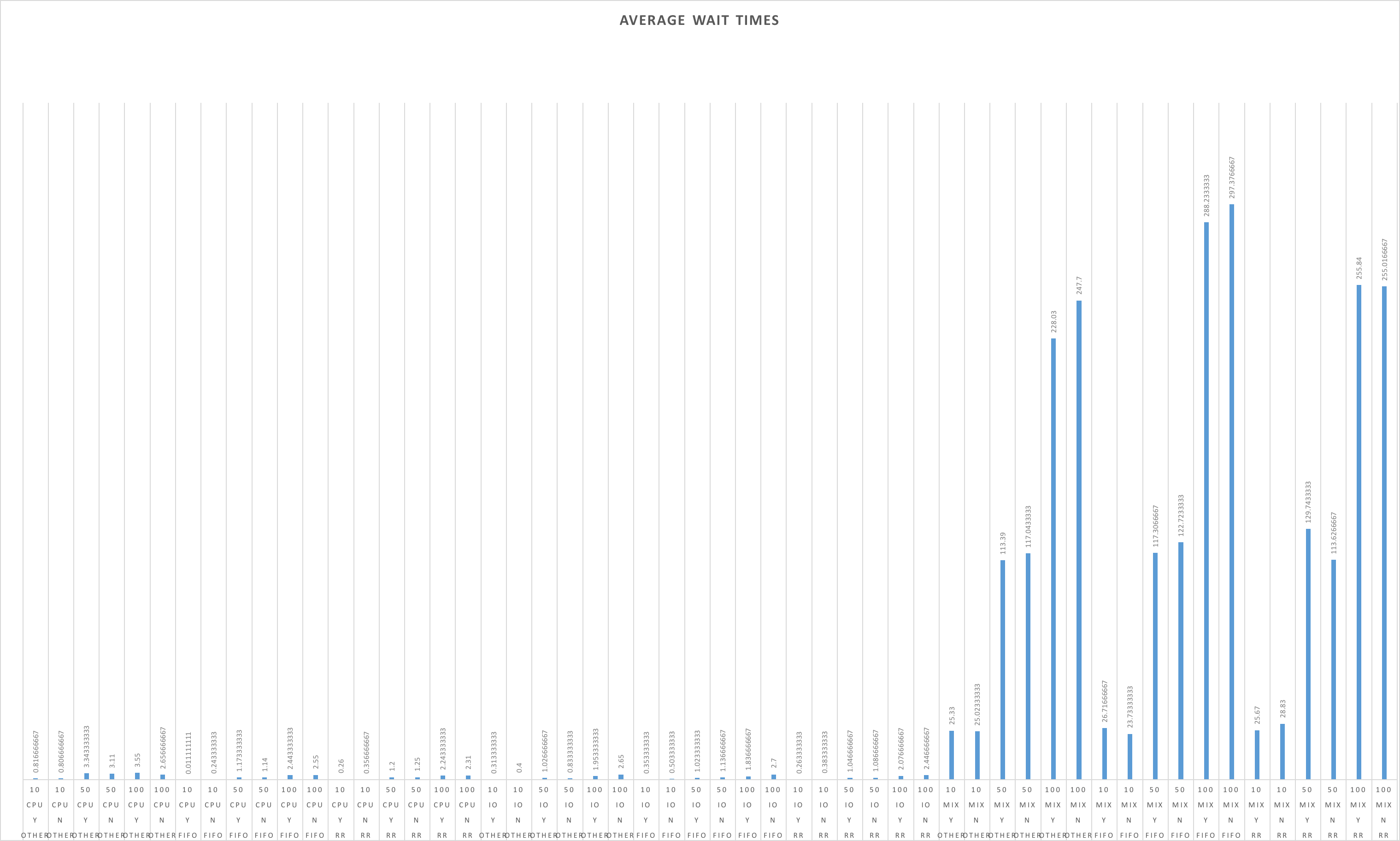
**Method:**

I created a single test file, *PA4\_test.c*, for my investigation that then called other programs that have CPU, IO, and mixed processes such as *pi.c*, *rw.c*,*and mixed.c*. My program takes in command line arguments that are the conditions in which we will be testing for. The first command argument it takes is the scheduler type. Second is the number of processes that scheduler will undergo, ranging from light (10), medium (50), and high (100). Third, the program takes in what type of process will be test a CPU, IO, or mixed. The final argument we receive is the priority and if it is uniform or not.

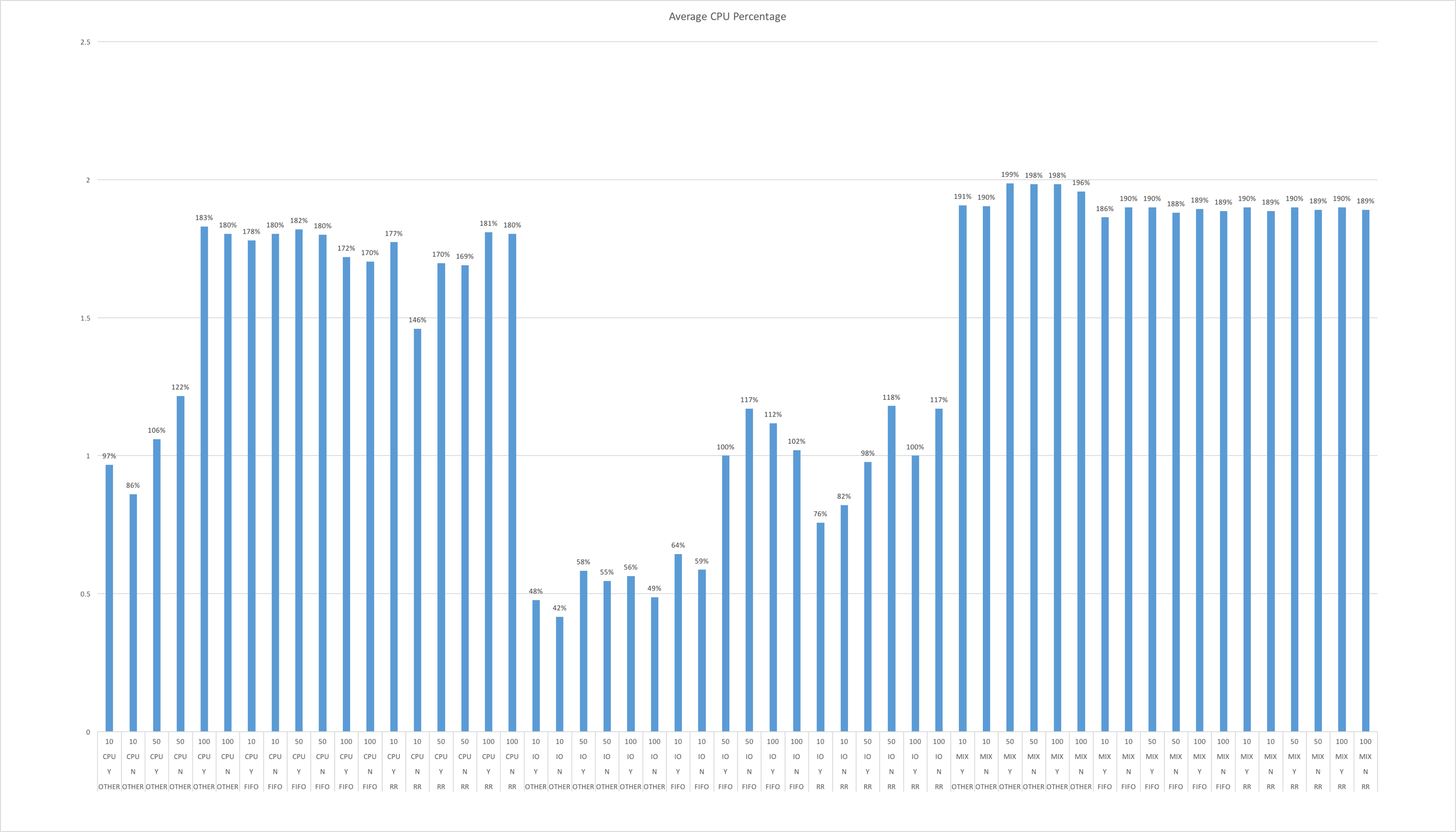
I wrote a shell script that runs my program calling every 54 possible conditions. Each set of conditions is looped through three times so I was able to average my data. The shell script then recorded the benchmarks to an output file where they could be analyzed in an excel worksheet. The benchmarks I put in place will help provide insight into which scheduler is best under the conditions given.

**Results:**

In *Figure 1* one we can see the various wall times for the tests. We can tell a few things from this chart. Most of the tests using the SCHED\_OTHER policy and had non-uniform nice values resulted in higher wall times. The other tests using SHED\_FIFO policy and non-uniform priority values resulted in longer wall times. The SCHED\_RR policy shows the least amount of change when it comes to wall times between uniform and non-uniform priority values.

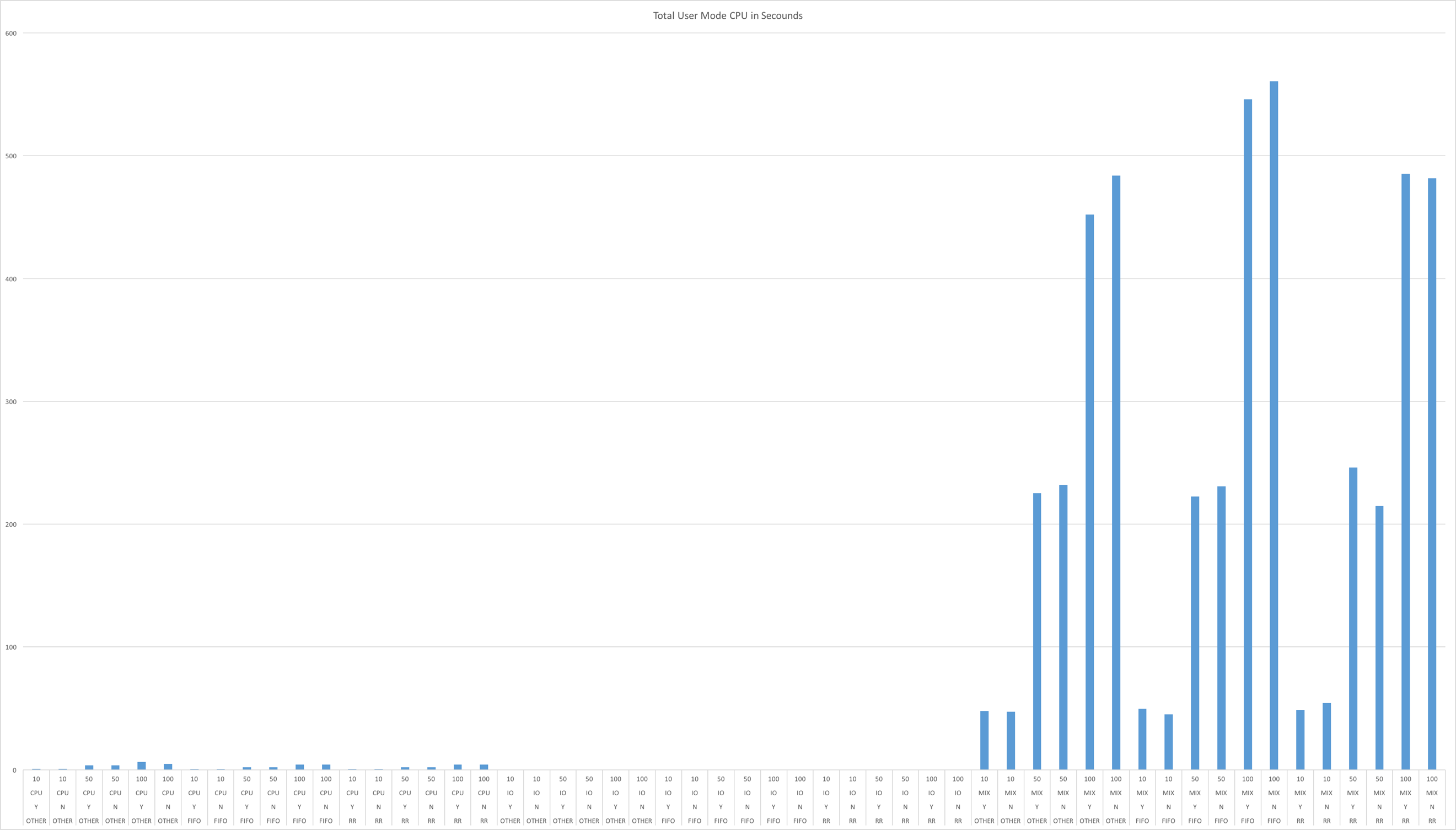


*Figure 1*

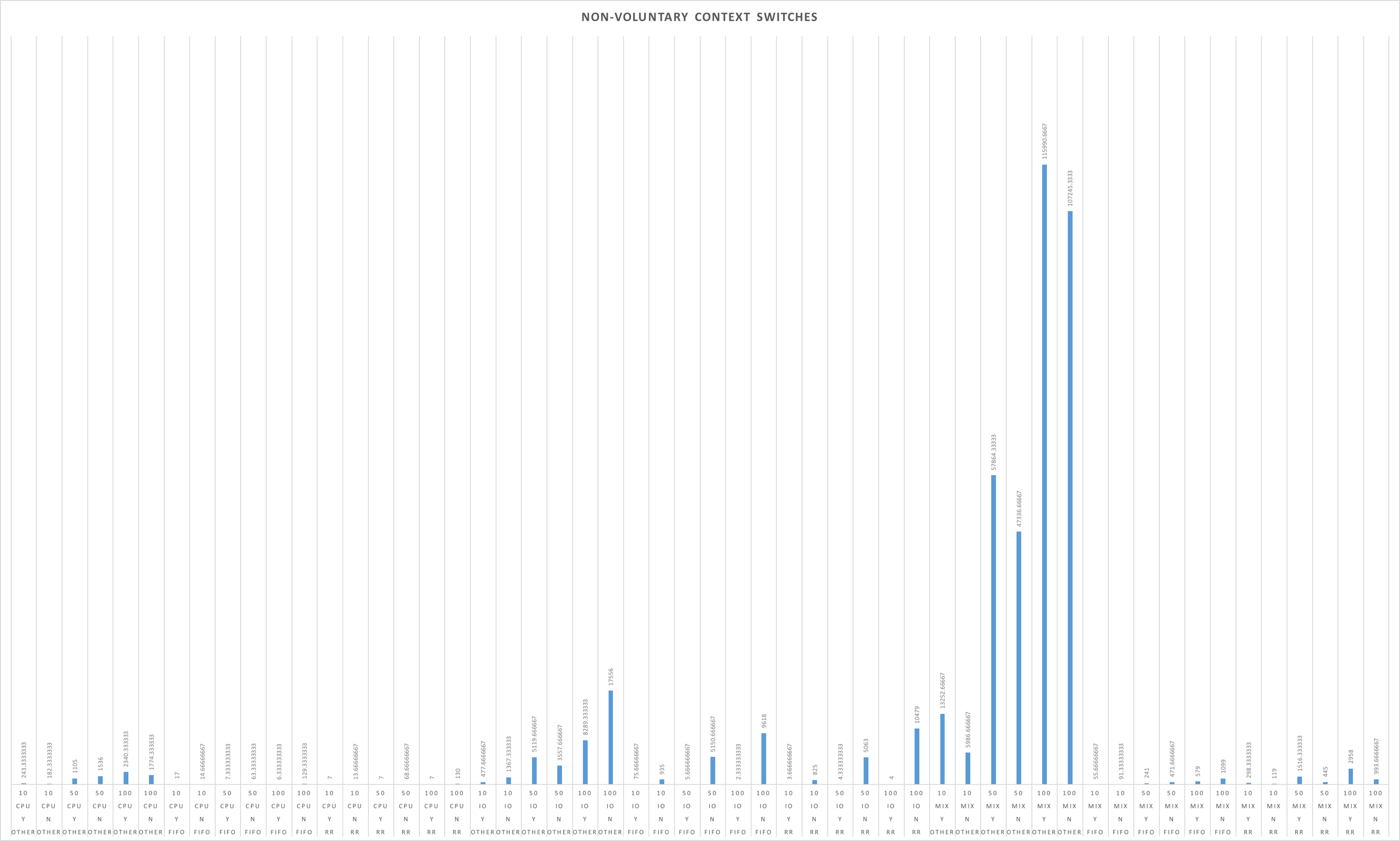


*Figure 2*

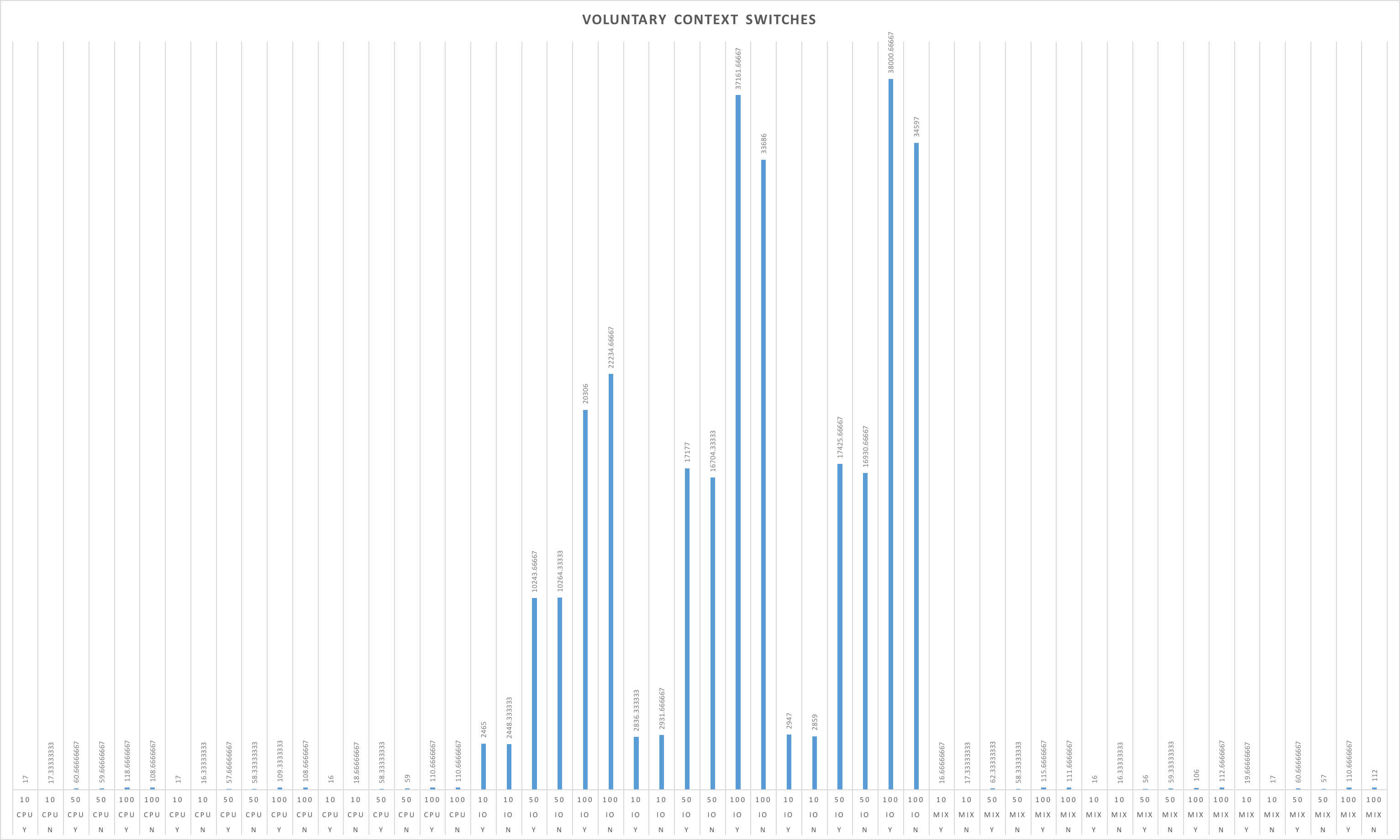
In *Figure 2* from are results we can generally see that the IO intensive operations did not use as much as the CPU which is expected. The data I collected did not vary much for the mix intensive operations however there is a pattern in the data. The non-uniform priority and nice values for SCHED\_OTHER and SCHED\_RR always resulted in a lower CPU percentage than uniformed values for mixed intensive processes. This pattern was also observed in the CPU intensive operations under the SCHED\_RR policy.



*Figure 3*

In *Figure 3* we can see that when the nice priority values are non-uniform there is more time spent in user mode using CPU. We can also see a vast difference in the SCHED\_FIFO policy for IO intensive processes. When there are non-uniform priority values there is many more non-voluntary context switches. 

*Figure 4*



*Figure 5*

**Analysis:**

After analyzing the data, I can speculate on some results about Linux schedulers. From my findings it seems that each scheduler has its conditions that it is best suited for. No one scheduler is the best across the board for all of the conditions I tested. However, it does seem that SCHED\_OTHER has some of the longer run times along with more overhead efficiency across all of the processes.

The SCHED\_RR policy seemed to be the best suit for CPU intensive processes from looking at the wall time however SCHED\_RR and SCHED\_FIFO used the CPU significantly more. The huge difference in the CPU percentages leads me to believe that even though the run time and overhead is larger for SCHED\_OTHER it uses much less of the CPU and therefore may be a valid scheduler for CPU intensive processes specially when there are fewer instances of the process. For I/O intensive processes SCHED\_RR and SCHED\_FIFO are very close in both run time and overhead efficiency. This could be the case because these schedulers are very similar and SCHED\_RR is a slight improvement to SCHED\_FIFO. The SCHED\_RR policy is the best for mix processes for a few reasons. It has some of the lowest wall times compared to the other scheduling policies while also keeping its overhead relatively low specially compared to SCHED\_OTHER. SCHED\_OTHER is the most inefficient policy for run time and overhead for mixed processes.

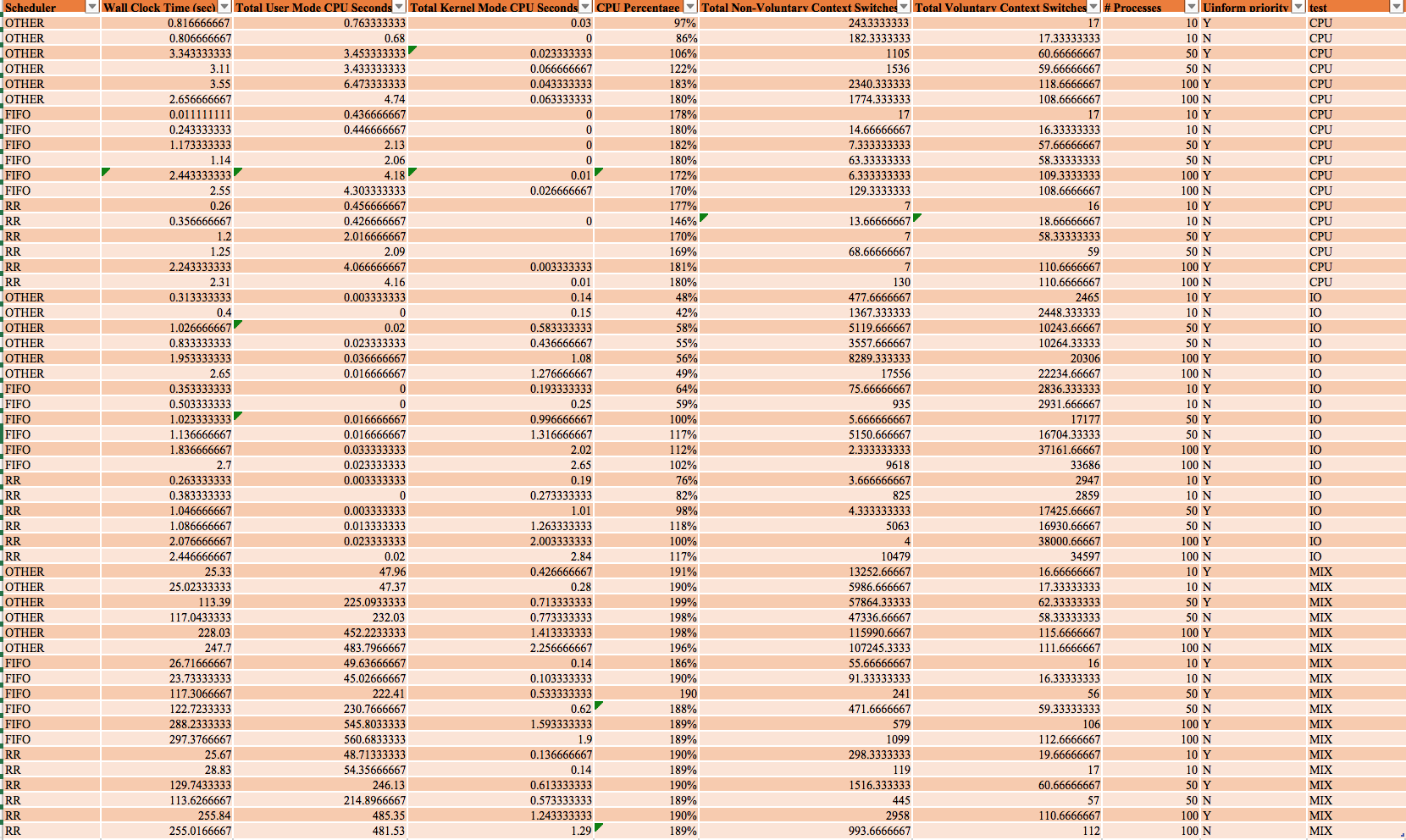
Like the conditions, each scheduler also scaled a little differently depending on the type of process. The SCHED\_OTHER does not scale well with I/O process and that leads to it not scaling well with the mixed processes either. SCHED\_OTHER has too much overhead to scale well with any of the processes. It does scale well with CPU intensive process though due to the variable-time slices the scheduler has. Both SCHED\_RR and SCHED\_FIFO scale fairly well for all of the processes. SCHED\_RR in specific has a low average wait time when process get length while also keeping shorter process efficient as well. It may not be the best for CPU intensive processes however these policies could be used in these cases.

**Conclusion:**

From my research and after analyzing my results I have come to some conclusions about the Linux schedulers. From my data SCHED\_OTHER is best suited in CPU intensive operations even though some of my data may argue that it has slightly higher run time and more overhead. This could be due to an error however the policy uses the least amount of CPU. From my data SCHED\_RR seems to be the best scheduler scaling well for all type of processes and number of instances. SCHED\_OTHER also scales fairly low having a fast run time and does not have a lot of overhead until mixed process. While these two are overall good SCHED\_FIFO does emerge as the best suited scheduler for I/O bound processes.

**References:**

**Appendix A:**



**Appendix B:**

*pi.c* – CPU intensive program

*rw.c* – I/O intensive program

*mixed.c* – CPU and IO intensive program

*HW4\_TEST.c* – main testing program

*Looptest.sh* – Shell script to run *HW4\_TEST.c* with all 54 tests and output to a file.

*Makefile* - The Makefile