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.

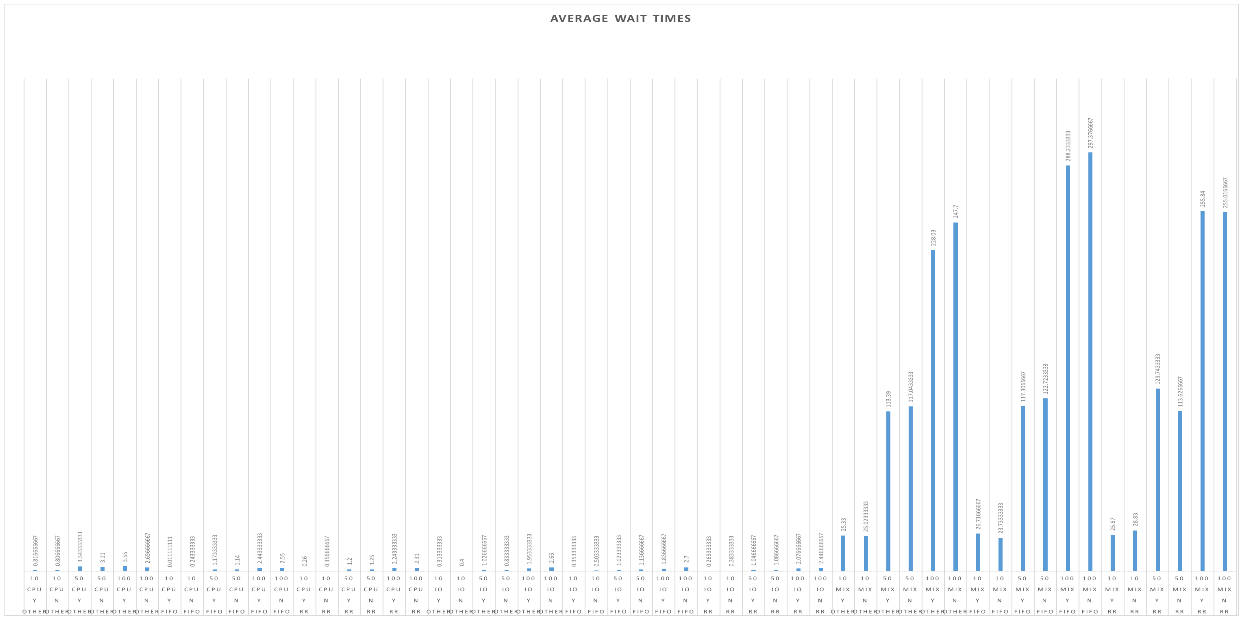


Figure 1

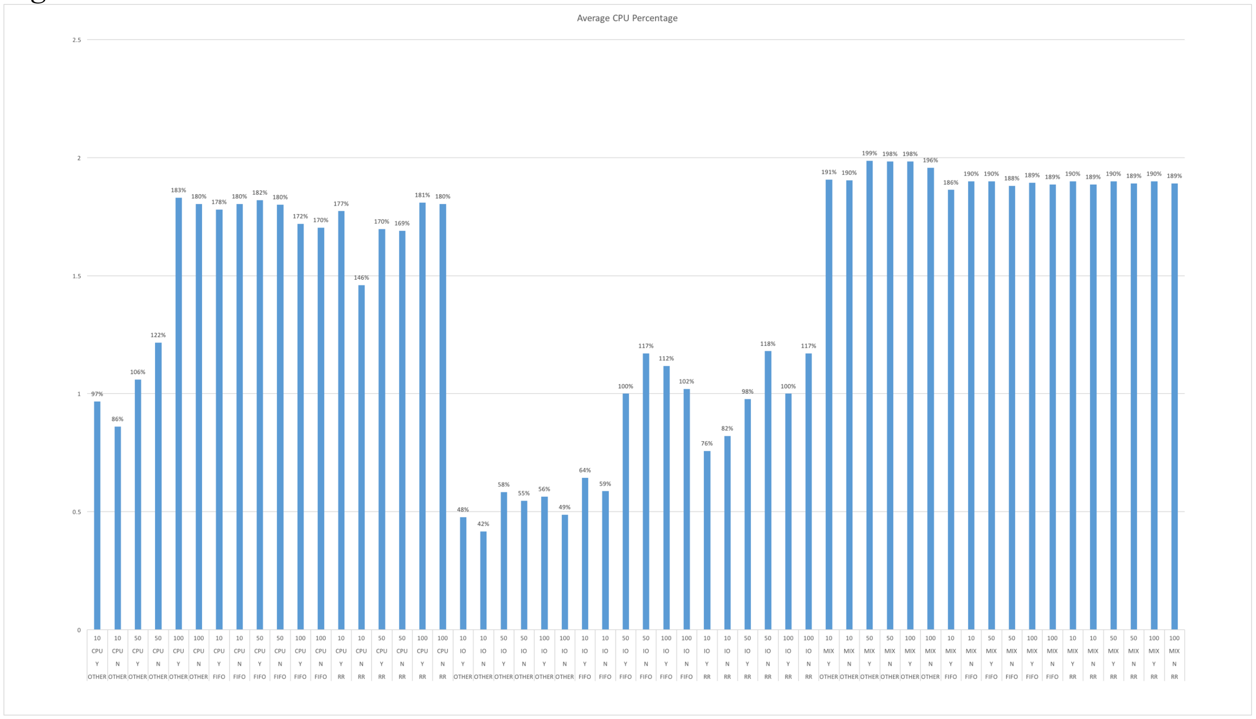


Figure 2

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.

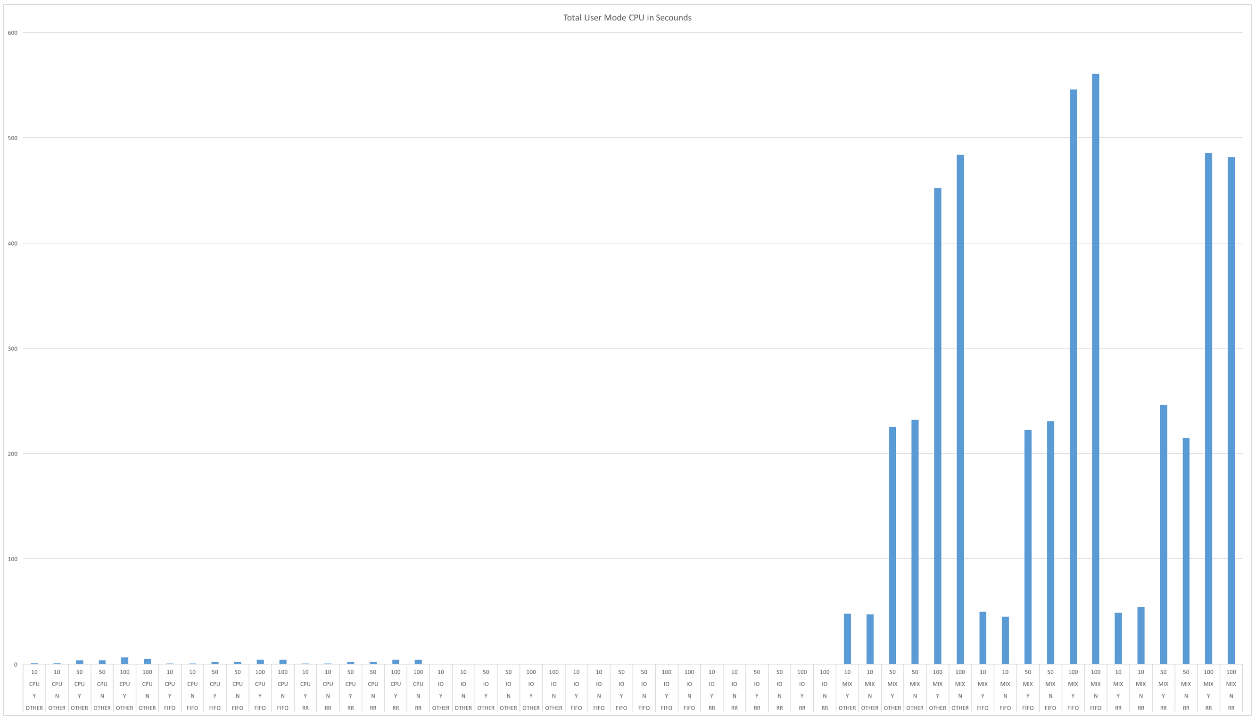


Figure 3

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.

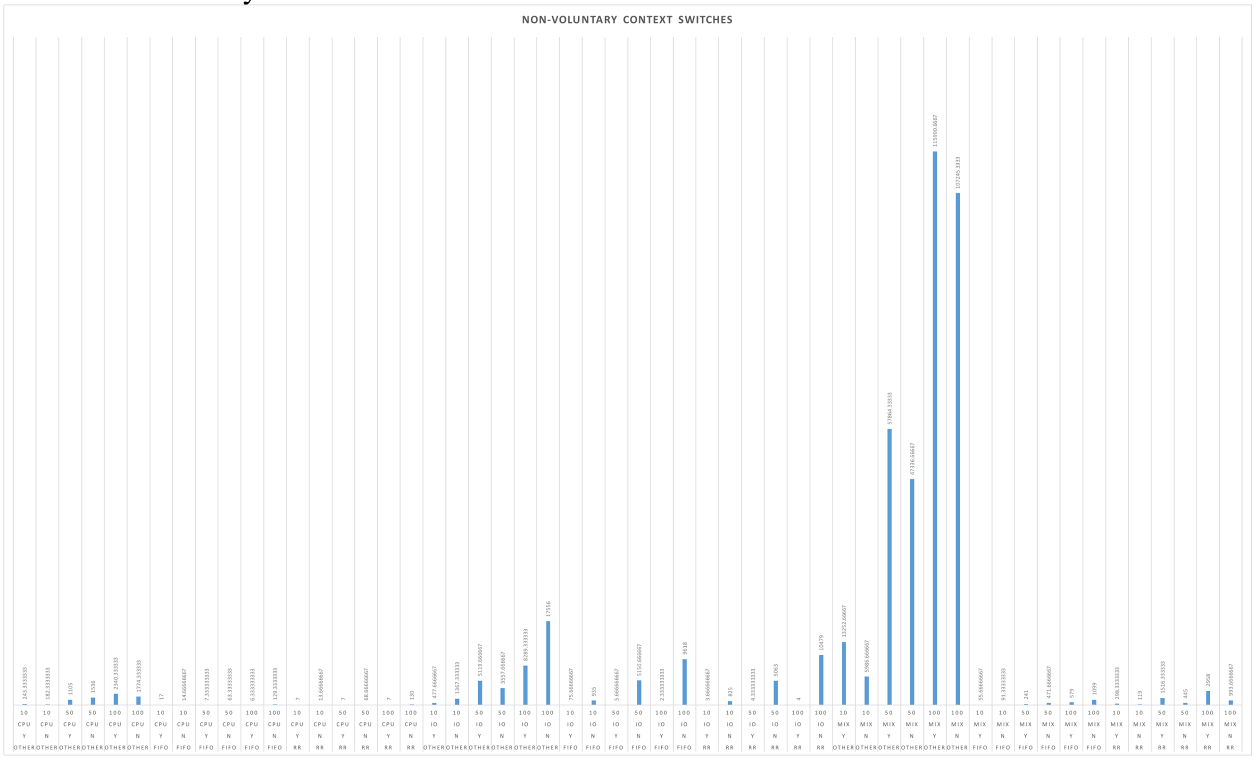


Figure 4

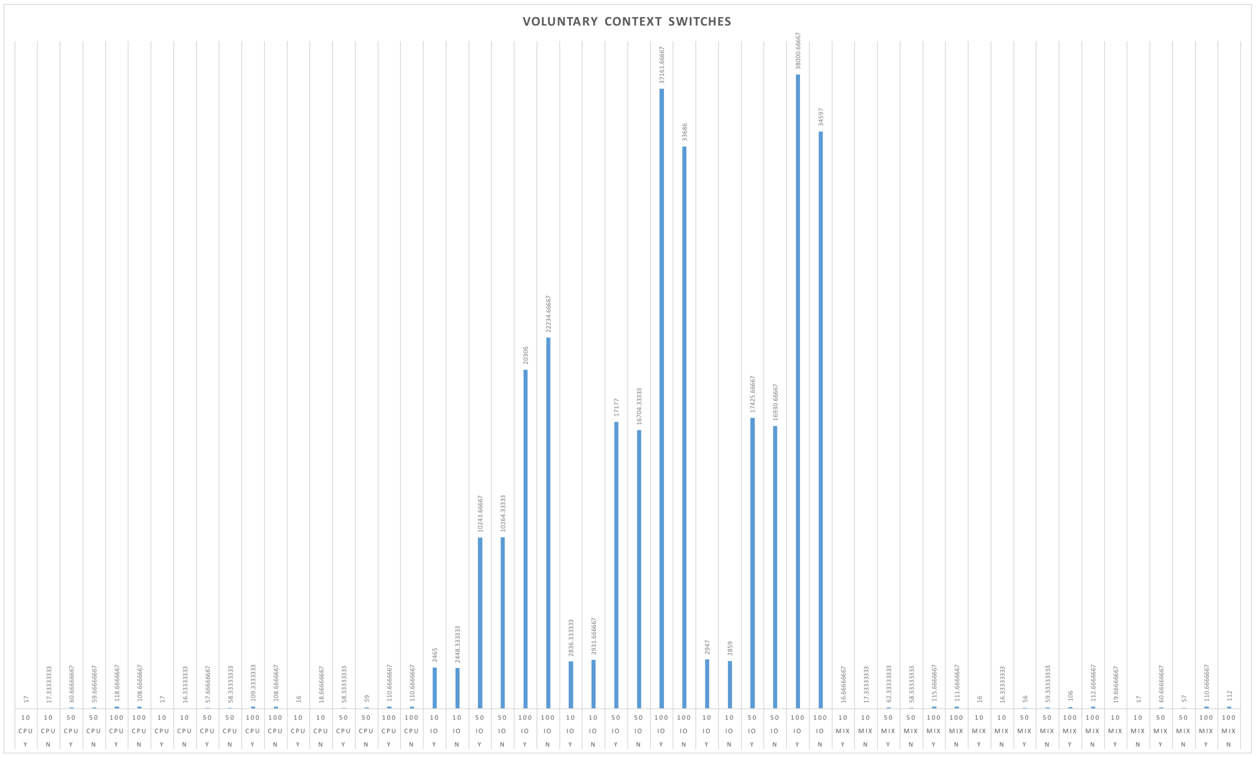


Figure 5

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.

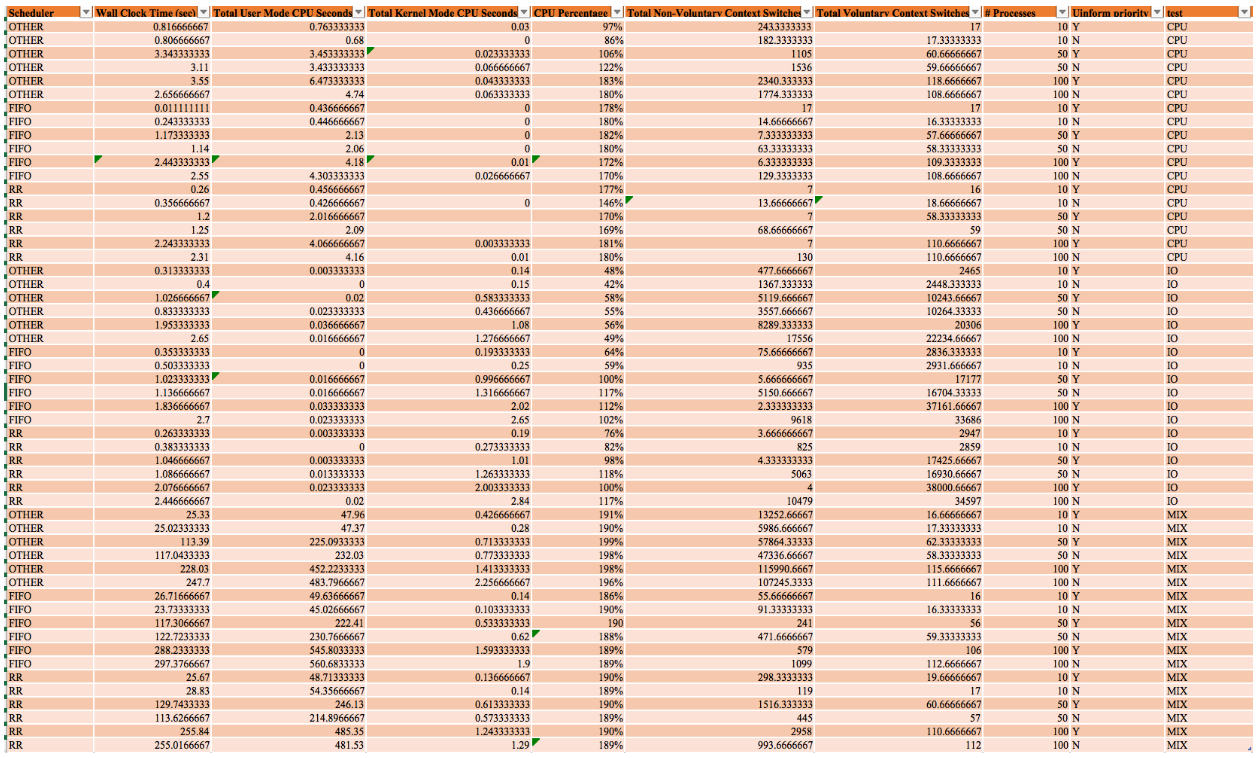
Like the conditions, each scheduler also scaled a little differently depending on the type of process. The SCHED_OTHER does not scale well with I/O process and that leads to it not scaling well with the mixed processes either. SCHED_OTHER has too much overhead to scale well with any of the processes. 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:



Scheduler	Wall Clock Time (sec)	Total User Mode CPU Seconds	Total Kernel Mode CPU Seconds	CPU Percentage	Total Non-Voluntary Context Switches	Total Voluntary Context Switches	# Processes	Uniform priority	test
OTHER	0.816666667	0.763333333	0.03	97%	243.3333333	17	10	Y	CPU
OTHER	0.806666667	0.68	0	86%	182.3333333	17.33333333	10	N	CPU
OTHER	3.343333333	3.453333333	0.023333333	106%	1105	60.66666667	50	Y	CPU
OTHER	3.11	3.433333333	0.066666667	122%	1536	59.66666667	50	N	CPU
OTHER	3.55	6.473333333	0.043333333	183%	2340.333333	118.6666667	100	Y	CPU
OTHER	2.656666667	4.74	0.063333333	180%	1774.333333	108.6666667	100	N	CPU
FIFO	0.011111111	0.436666667	0	178%	17	17	10	Y	CPU
FIFO	0.243333333	0.446666667	0	180%	14.66666667	16.33333333	10	N	CPU
FIFO	1.173333333	2.13	0	182%	7.333333333	57.66666667	50	Y	CPU
FIFO	1.14	2.06	0	180%	63.33333333	58.33333333	50	N	CPU
FIFO	2.443333333	4.18	0.01	172%	6.333333333	109.3333333	100	Y	CPU
FIFO	2.55	4.303333333	0.026666667	170%	129.3333333	108.6666667	100	N	CPU
RR	0.26	0.456666667		177%	7	16	10	Y	CPU
RR	0.356666667	0.426666667	0	146%	13.66666667	18.66666667	10	N	CPU
RR	1.2	2.016666667		170%	7	58.33333333	50	Y	CPU
RR	1.25	2.09		169%	68.66666667	59	50	N	CPU
RR	2.243333333	4.066666667	0.003333333	181%	7	110.6666667	100	Y	CPU
RR	2.31	4.16	0.01	180%	130	110.6666667	100	N	CPU
OTHER	0.313333333	0.003333333	0.14	48%	477.6666667	2465	10	Y	IO
OTHER	0.4	0	0.15	42%	1367.333333	2448.333333	10	N	IO
OTHER	1.026666667	0.02	0.583333333	58%	5119.666667	10243.66667	50	Y	IO
OTHER	0.833333333	0.023333333	0.436666667	55%	3557.666667	10264.33333	50	N	IO
OTHER	1.953333333	0.036666667	1.08	56%	8289.333333	20306	100	Y	IO
OTHER	2.65	0.016666667	1.276666667	49%	17556	22234.66667	100	N	IO
FIFO	0.353333333	0	0.193333333	64%	75.66666667	2836.333333	10	Y	IO
FIFO	0.503333333	0	0.25	59%	935	2931.666667	10	N	IO
FIFO	1.023333333	0.016666667	0.996666667	100%	5.666666667	17177	50	Y	IO
FIFO	1.136666667	0.016666667	1.316666667	117%	5150.666667	16704.33333	50	N	IO
FIFO	1.836666667	0.033333333	2.02	112%	2.333333333	37161.66667	100	Y	IO
FIFO	2.7	0.023333333	2.65	102%	9618	33686	100	N	IO
RR	0.263333333	0.003333333	0.19	76%	3.666666667	2947	10	Y	IO
RR	0.383333333	0	0.273333333	82%	825	2859	10	N	IO
RR	1.046666667	0.003333333	1.01	98%	4.333333333	17425.66667	50	Y	IO
RR	1.086666667	0.013333333	1.263333333	118%	5063	16930.66667	50	N	IO
RR	2.076666667	0.023333333	2.003333333	100%	4	38000.66667	100	Y	IO
RR	2.446666667	0.02	2.84	117%	10479	34597	100	N	IO
OTHER	25.33	47.96	0.426666667	191%	13252.66667	16.66666667	10	Y	MIX
OTHER	25.02333333	47.37	0.28	190%	5986.666667	17.33333333	10	N	MIX
OTHER	113.39	225.0933333	0.713333333	199%	57864.33333	62.33333333	50	Y	MIX
OTHER	117.0433333	232.03	0.773333333	198%	47336.66667	58.33333333	50	N	MIX
OTHER	228.03	452.2233333	1.413333333	198%	115990.6667	115.6666667	100	Y	MIX
OTHER	247.7	483.7966667	2.256666667	196%	107245.3333	111.6666667	100	N	MIX
FIFO	26.71666667	49.63666667	0.14	186%	55.66666667	16	10	Y	MIX
FIFO	23.73333333	45.02666667	0.103333333	190%	91.33333333	16.33333333	10	N	MIX
FIFO	117.3066667	222.41	0.533333333	190	241	56	50	Y	MIX
FIFO	122.7233333	230.7666667	0.62	188%	471.6666667	59.33333333	50	N	MIX
FIFO	288.2333333	545.8033333	1.593333333	189%	579	106	100	Y	MIX
FIFO	297.3766667	560.6833333	1.9	189%	1099	112.6666667	100	N	MIX
RR	25.67	48.71333333	0.136666667	190%	298.3333333	19.66666667	10	Y	MIX
RR	28.83	54.35666667	0.14	189%	119	17	10	N	MIX
RR	129.7433333	246.13	0.613333333	190%	1516.333333	60.66666667	50	Y	MIX
RR	113.6266667	214.8966667	0.573333333	189%	445	57	50	N	MIX
RR	255.84	485.35	1.243333333	190%	2958	110.6666667	100	Y	MIX
RR	255.0166667	481.53	1.29	189%	993.6666667	112	100	N	MIX

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