CSCI 3753 (Operating Systems)
University of Colorado at Boulder
Spring 2014
Programming Assignment 3
Sahle Alturaigi

Abstract

This report is a short analysis and summary of all the work done in programming assignment 3. In this programming assignment, some scheduling policies were benchmarked with different types of programs to evaluate which policy is most efficient in some aspects.

The machine that was used for this experiment was an I7 quad core Alienware m15x 2011 model. The tests and results indicated that SCHED_OTHER was the always the quickest policy for running programs. The only thing notable about RT policies that were discovered in this experiment was that they conducted much fewer context switches than SCHED_OTHER. If power consumption is an issue with a platform, RT scheduling policies should be considered, otherwise SCHED_OTHER would be recommended.

Introduction

The goal of this programming assignment was to benchmark how several different Linux scheduling policies performed with CPU bound programs, I/O bound programs and a combination of CPU and I/O bound programs. Linux has many scheduling policies, however, the policies that were used for this benchmark were SCHED_OTHER (or SCHED_NORMAL), SCHED_RR and SCHED_FIFO. SCHED_OTHER is a CFS policy that time-slices each process so that they all have a certain amount of time to use the CPU. SCHED_RR is a round-robin scheduling policy and SCHED_FIFO is a first-in-first-out policy. Each other them have their pros and cons and this report will go into detail where each policy is appropriate to use. In this report, the reader is assumed to have some understanding of how scheduling policies work, what benchmarking is, and how to interpret benchmarked data that is related to computer performance.

Method

In this programming assignment, I used three different programs to test three different scheduling policies with three different numbers of processes. The policies that were used are mentioned above. The three types of programs are:

1. CPU Bound
2. I/O Bound
3. CPU and I/O Bound

The three numbers of processes generated for each test are categorized in the following way:

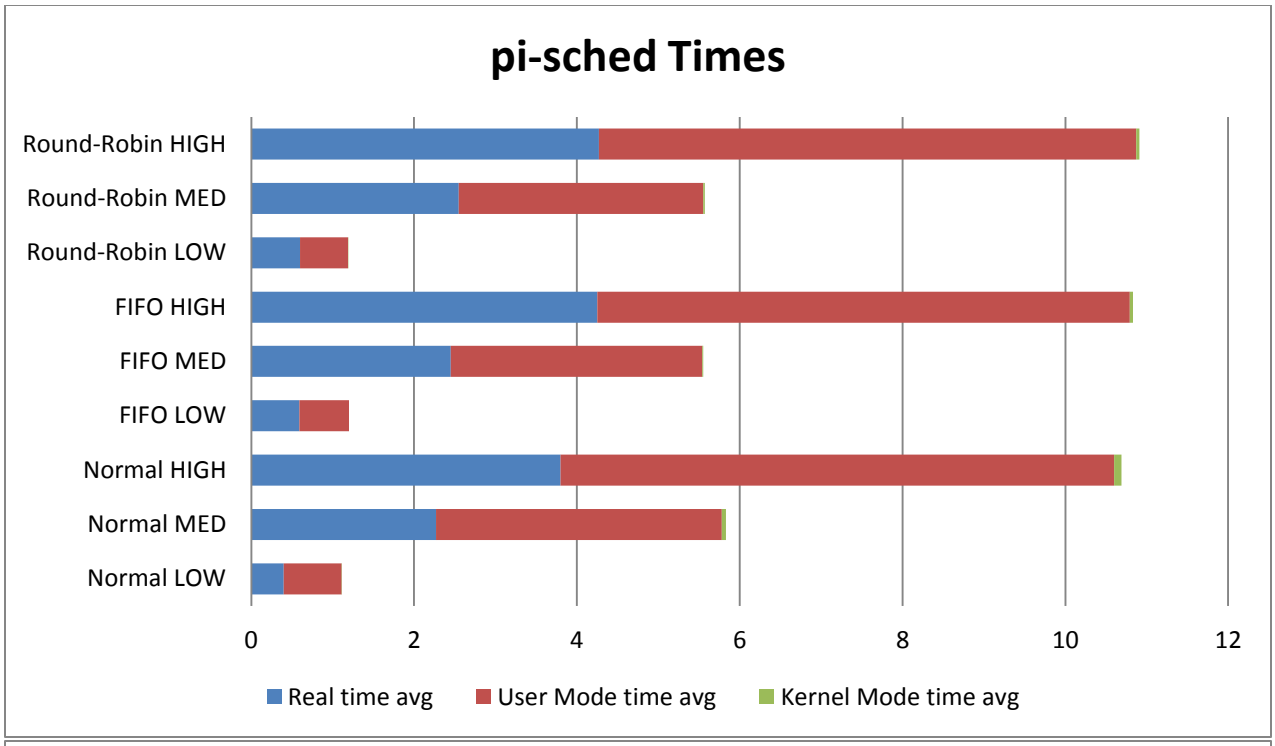
1. LOW (10 processes)
2. MEDIUM (50 processes)
3. HIGH (100 processes)

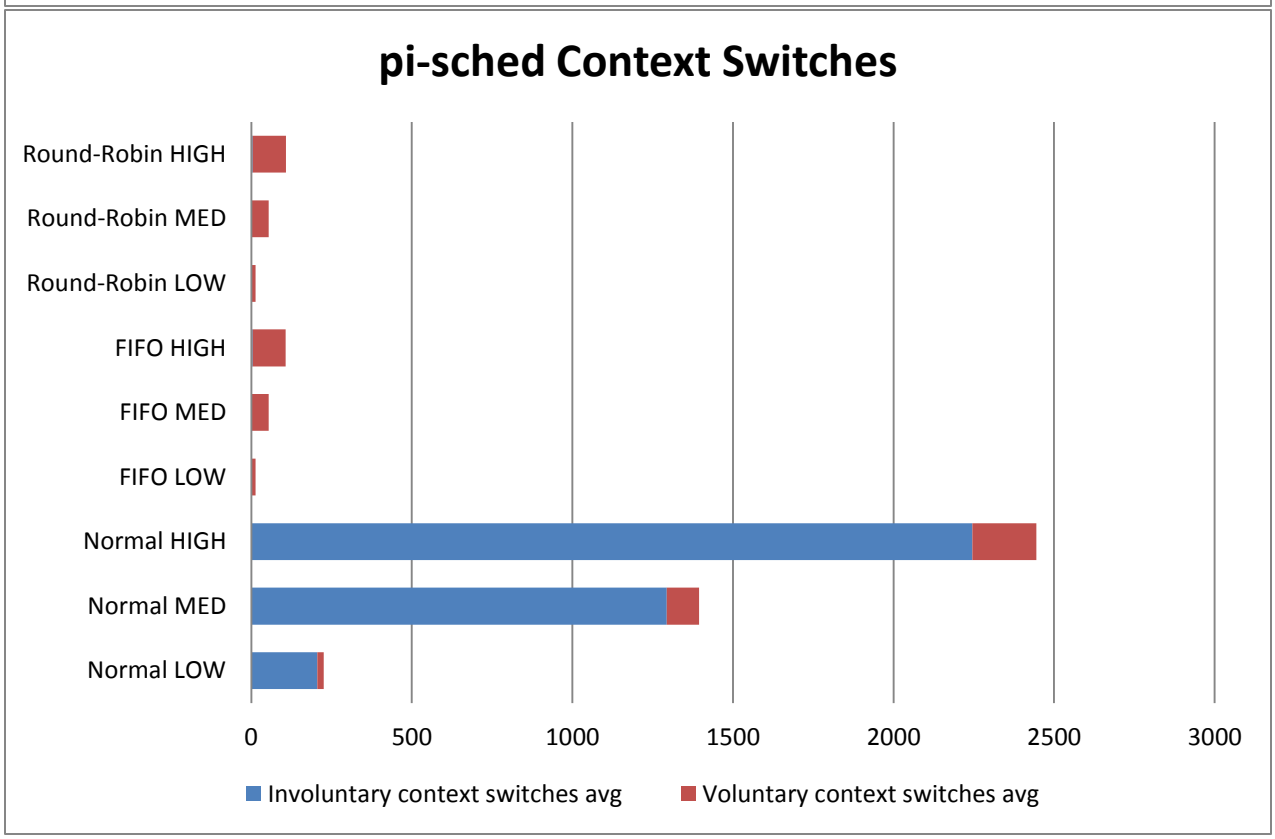
All of the programs in this report were structured in the fashion that they asked for a scheduling policy (if none was supplied, use SCHED_OTHER) and how many processes to run (if none was supplied, use LOW). Once the aforementioned completed, each program applied their policy and then forked N number (supplied by user) of children. Each child process ran their own program, which was defined above, and then terminated once it completed.

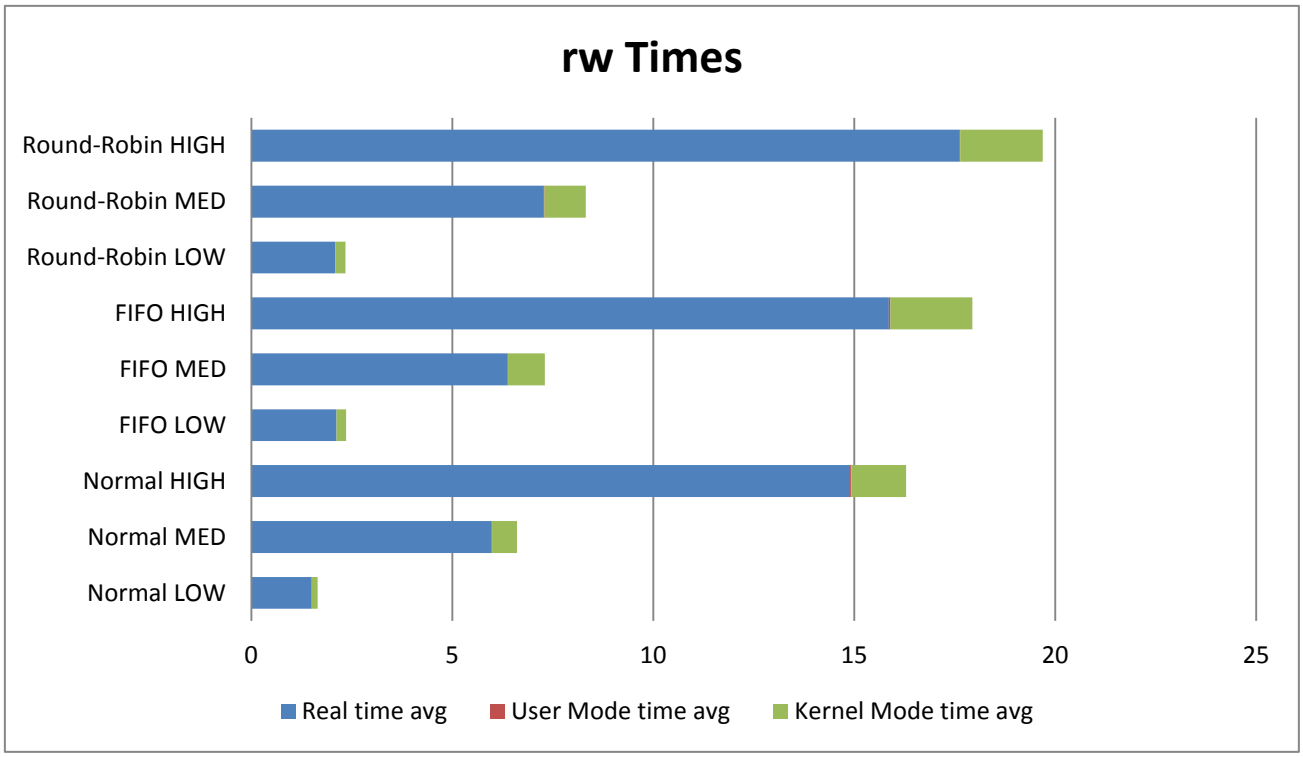
A bash script was utilized to gather and output all the benchmark data into separate files for all the tests that were completed. A total of 27 unique tests were completed with 7 tests within each unique test such that the data could be averaged and that error margin from wild results is reduced. All of the raw data can be found in Appendix A for your reference.

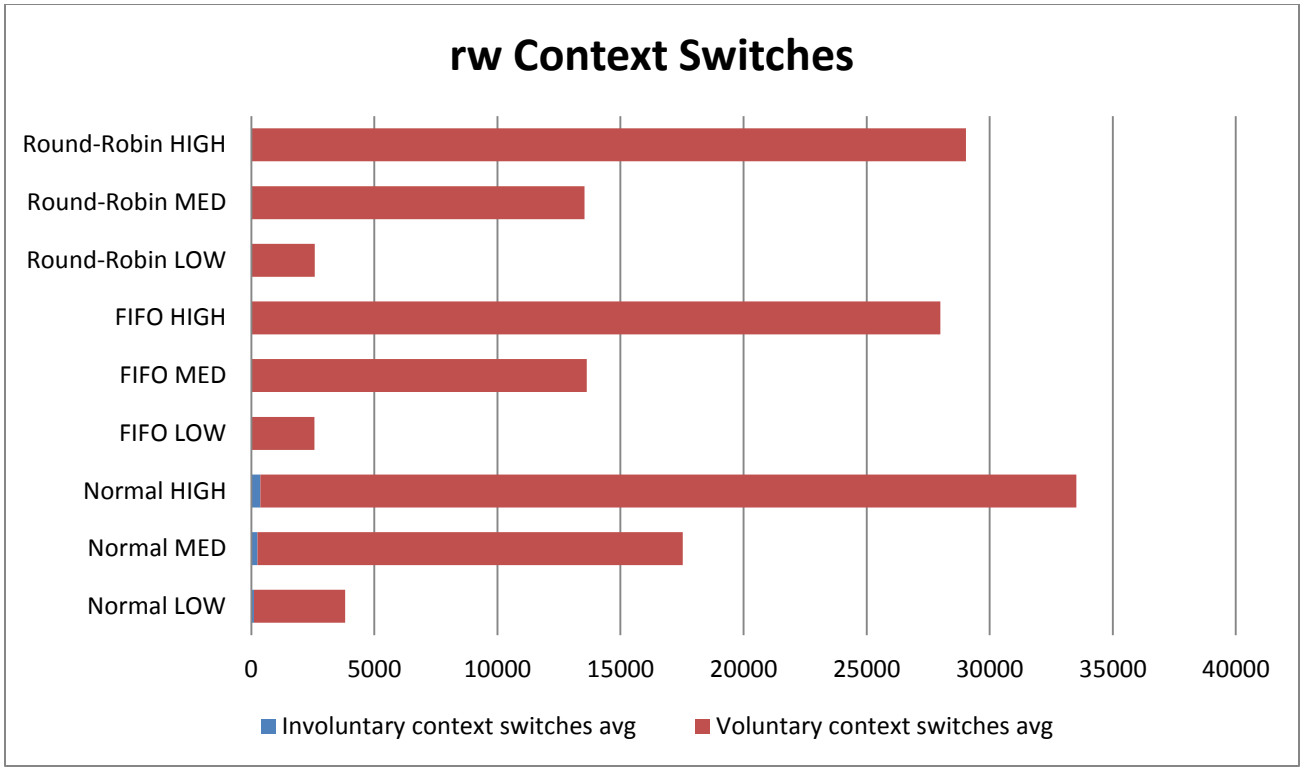
Results

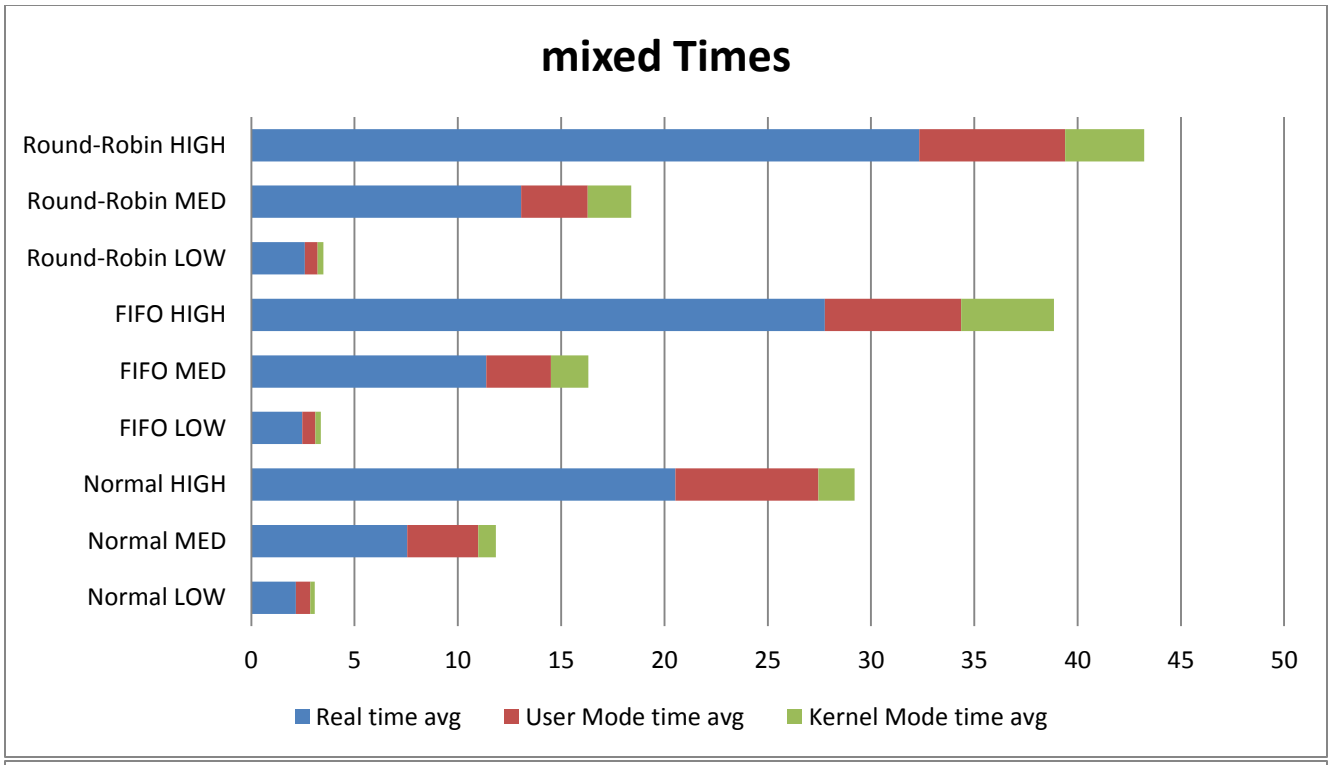
In this section, the data from bash scripts has been organized into bar charts. Each page in the results section below has two bar charts showing the results for time and context switches when using different policies and different number of simultaneously running processes. The time charts are in the units of seconds. After studying the data for a short moment, there are noticeable trends that can be seen.

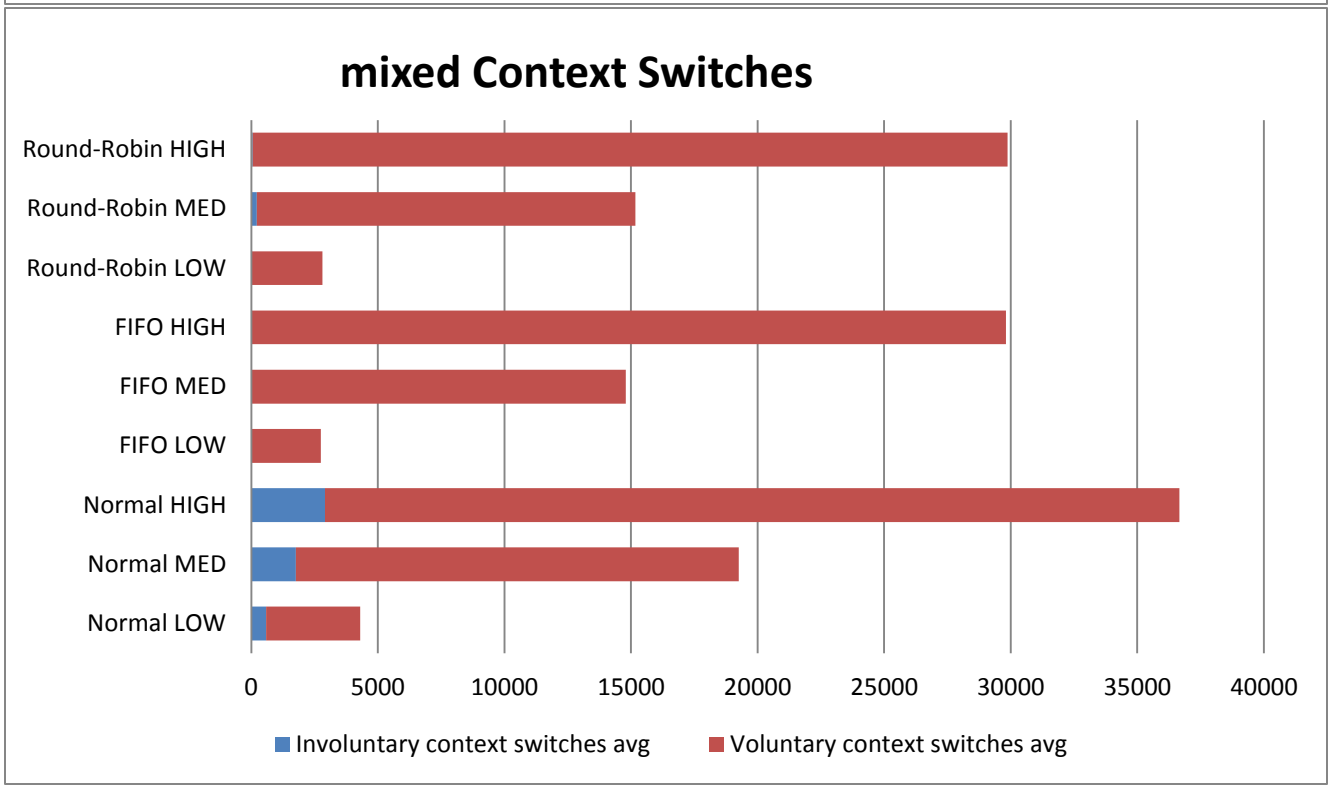

pi-sched Times
Round-Robin HIGH
Round-Robin MED
Round-Robin LOW
FIFO HIGH
FIFO MED
FIFO LOW
Normal HIGH
Normal MED
Normal LOW
0
2
4
6
8
10
12
Real time avg
User Mode time avg
Kernel Mode time avg


pi-sched Context Switches
Round-Robin HIGH
Round-Robin MED
Round-Robin LOW
FIFO HIGH
FIFO MED
FIFO LOW
Normal HIGH
Normal MED
Normal LOW
0
500
1000
1500
2000
2500
3000
Involuntary context switches avg
Voluntary context switches avg


rw Times
Round-Robin HIGH
Round-Robin MED
Round-Robin LOW
FIFO HIGH
FIFO MED
FIFO LOW
Normal HIGH
Normal MED
Normal LOW
0
5
10
15
20
25
Real time avg
User Mode time avg
Kernel Mode time avg


rw Context Switches
Round-Robin HIGH
Round-Robin MED
Round-Robin LOW
FIFO HIGH
FIFO MED
FIFO LOW
Normal HIGH
Normal MED
Normal LOW
0
5000
10000
15000
20000
25000
30000
35000
40000
Involuntary context switches avg
Voluntary context switches avg


mixed Times
Round-Robin HIGH
Round-Robin MED
Round-Robin LOW
FIFO HIGH
FIFO MED
FIFO LOW
Normal HIGH
Normal MED
Normal LOW
0
5
10
15
20
25
30
35
40
45
50
Real time avg
User Mode time avg
Kernel Mode time avg


mixed Context Switches
Round-Robin HIGH
Round-Robin MED
Round-Robin LOW
FIFO HIGH
FIFO MED
FIFO LOW
Normal HIGH
Normal MED
Normal LOW
0
5000
10000
15000
20000
25000
30000
35000
40000
Involuntary context switches avg
Voluntary context switches avg

Analysis and Conclusion

The data found above is exactly the same as the raw data found in appendix A, only reformatted for readability. Due to the very small difference there was in terms of how much CPU each policy used, I will omit this from the analysis since drawing any conclusions from this would require much more thorough tests. The analysis will be done for each program first and then aggregately. Pi-sched is a CPU intensive program; this will be analyzed first. Both of the Real-Time processes, which are round-robin and FIFO, performed almost identically in terms of context switches and run-time in pi-sched. From the results, FIFO and RR take the least amount of context switches, yet ironically also take the most time to complete. OTHER takes the most context switches and the least amount of time to complete a program. As a programmer, if context switches are a sensitive issue then using a RT policy would be preferred over OTHER. If performance is an issue, then OTHER would be the best policy for completing the program fastest. On a theoretical level this does not make full sense and therefore there must be other factors that are not being considered (i.e. hardware or pipelining). One possible conclusion to this outcome is that the hardware designed for this computer makes it so that the SCHED_OTHER policy may work more efficiently that what it is normally supposed to be. Hardware such as the MMU (memory management units) could be the factor here causing a shift in the results.

In the I/O bound program test, there was almost an exact amount of voluntary context switches in all policies, with SCHED_OTHER having a few more voluntary context switches. Having said that, the run-time performance varied greatly with RR being the slowest, FIFO in the middle, and OTHER being the fastest. It seems that for this machine (I7 quad core processor) the OTHER scheduling policy works best.

Lastly, in a program with both CPU and I/O bound code, the results varied just like the I/O bound test program. RR performed the worst and OTHER performed the best in terms of run-time. As for voluntary context switches, the results are very similar to I/O bound, again, where FIFO and RR had similar results, but OTHER had more context switches.

To conclude this analysis, on this machine (I7 quad core, Alienware m15x 2011 model) the SCHED_OTHER scheduling policy is the most efficient relative to run-time. This is probably why the Linux OS uses SCHED_OTHER as the default policy. The only thing that SCHED_OTHER struggled in compared to the RT policies like FIFO and RR were context switches. Context switches cause overhead and use processing power. For platforms where energy conservation is an issue, like mobile phones, RT scheduling policies may be preferred in order to extend battery life.

References

[1] S. Galvin, Opertaing Systems 8th edition.

Appendix A

pi-sched Process	Normal LOW	Normal MED	Normal HIGH	FIFO LOW	FIFO MED	FIFO HIGH	Round-Robin LOW	Round-Robin MED	Round-Robin HIGH
Real time avg	0.4	2.27	3.8	0.59	2.45	4.25	0.6	2.55	4.27
User Mode time avg	0.71	3.51	6.8	0.61	3.09	6.54	0.59	3	6.6
Kernel Mode time avg	0.004	0.05	0.09	0	0.01	0.04	0.001	0.02	0.04
% CPU used avg	1.8	1.57	1.82	1.04	1.31	1.54	0.1	1.23	1.56
Involuntary context switches avg	205.14	1293.86	2245.43	1	1	4.29	0.86	1.43	4.43
Voluntary context switches avg	20.86	100.14	199.14	12.29	52.86	102.86	12.14	52.71	103

rw Process	Normal LOW	Normal MED	Normal HIGH	FIFO LOW	FIFO MED	FIFO HIGH	Round-Robin LOW	Round-Robin MED	Round-Robin HIGH
Real time avg	1.5	5.97	14.9	2.12	6.377	15.86	2.09	7.28	17.61
User Mode time avg	0.003	0.02	0.03	0	0.007	0.04	0	0.015	0.03
Kernel Mode time avg	0.15	0.62	1.36	0.24	0.92	2.04	0.25	1.03	2.05
% CPU used avg	0.1	0.11	0.09	0.11	0.14	0.12	0.12	0.14	0.11
Involuntary context switches avg	102	249.86	373.29	0.43	0.86	0.57	0.57	0.57	0.42
Voluntary context switches avg	3707.14	17274	33148	2569.57	13626.7	27988.1	2581.57	13536	29031

mixed Process	Normal LOW	Normal MED	Normal HIGH	FIFO LOW	FIFO MED	FIFO HIGH	Round-Robin LOW	Round-Robin MED	Round-Robin HIGH
Real time avg	2.15	7.55	20.53	2.47	11.38	27.76	2.59	13.07	32.34
User Mode time avg	0.7	3.44	6.92	0.63	3.12	6.6	0.62	3.21	7.06
Kernel Mode time avg	0.22	0.85	1.76	0.27	1.81	4.49	0.28	2.11	3.83
% CPU used avg	0.43	0.588	0.42	0.37	0.44	0.4	0.35	0.41	0.34
Involuntary context switches avg	586.71	1761.85	2914.29	0.57	1	3.57	1.29	221.29	53.57
Voluntary context switches avg	3710.71	17496	33752.3	2740.71	14798.3	29804	2807.71	14957.7	29815.9

Appendix B

1. **pi-sched.c** Code for running a statistically running pi with a specific scheduling policy.

2. **rw.c** A small I/O bound program that will copy N bytes from a default input file to a default output file using a specific scheduling policy.

3. **mixed.c** This file is a combination of both pi-sched.c and rw.c running with a specific scheduling policy specified from the user.

4. **testscript** This is a script that compiles all the code, runs it, and then outputs all the collected data into the file /test_output.

5. **testscript2** This is an updated version of my previous test script used for outputting the data into a particular format so that I could make charts and graphs.

6. **README** Contains directions on how to use these programs in detail.

7. **/test_output/** The destination folder for all the outputs generated from testscript and testscript2.