William Christie
SID 810915676
CSCI 3753-103 Fall, 2016
Programming Assignment 4 Write Up

Abstract

This paper explores the differences between the SCHED_OTHER (CFS) policy of the CFS scheduler class and the real time scheduler class in Linux consisting of SCHED_FIFO (first-in-first-out) and SCHED_RR (round robin). In order to collect data to draw comparisons, three test programs were written to model the behavior of a CPU-bound, IO-bound and mixed behavior processes. Additionally, a bash testscript was written to run fifty-four test vectors to examine the three processes and how they scale at certain priority levels and increasing numbers of simultaneous processes. This paper then analyzes the relevant data collected from running these test vectors and seeks to draw conclusions regarding which process types is most suitable to which scaling policy, how the different policies scale, provide pros and cons of the three scheduling policies, and provide general examples of instances where certain type of process would benefit from a certain scheduling policy and instances where a certain policy would not benefit a certain type of process.

Introduction

The focus of this experiment was to learn about the differences between the SCHED_OTHER (CFS) policy of the CFS scheduler class and the SCHED_FIFO (first-in-first-out) and SCHED_RR (round robin) policies of the real time scheduler class in Linux. In investigating these three policies I was able to determine which scheduling policy is best suited for CPU-bound, I/O-bound, and mixed process types in terms of overhead efficiency and runtime, how each scheduling policy scales at LOW, MEDIUM, and HIGH numbers of simultaneous processes, how each policy differs when the priorities of the simultaneous processes are different or the same, and what pros and cons exist with each scheduling policy. In order to answer the above questions it was necessary to create test programs to evaluate fifty-four test vectors using the method described below.

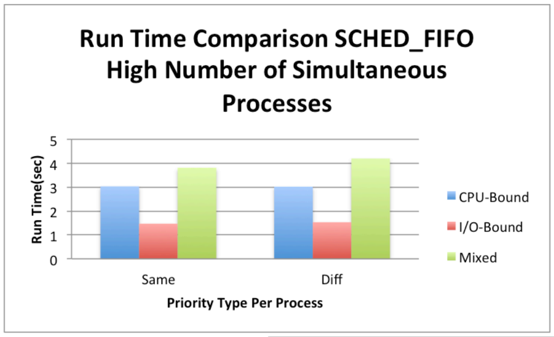
Method

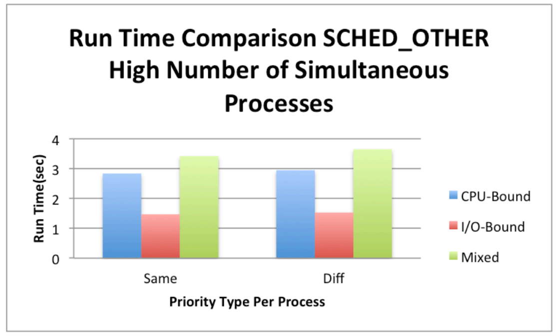
Three test programs were written in order to obtain benchmarks for the three different process types. The first program, `cpu-bound.c`, is a CPU-bound program that calculates the value of pi over one million iterations. The second program, `io-bound.c` is an I/O-bound program which reads a specified number of bytes from an input file to an output files. The final program, `mixed.c`, performs a mixture of CPU-bound and I/O operations by calculating the value of pi over one hundred thousand iterations as in `cpu-bound.c`, stopping at every one thousandth iteration in order to perform the read and write operation as described in `io-bound.c`. `Cpu-bound.c` and `io-bound.c` were adapted from `pi-sched.c` and `rw.c` that were written by Sayler and Mishra (2016) and provided as jumping off points for the experiment. `Mixed.c` was then written as a simple mixture of `cpu-bound.c` and `io-bound.c`. These three programs take as inputs a scheduling policy, a number of simultaneous processes (LOW=10, MEDIUM=50, HIGH=150) that the program should make using the fork command, and an indicator of whether the simultaneous processes should have the same or different niceness/priority per process. The children processes are forked to execute the specific CPU-bound or I/O-bound instructions while the parent process waits for and reaps the children once they finish execution. In the case where niceness/priority values should be the same, all processes were set to the priority indicated by sched_get_priority_max function for the specific policy. In contrast, in the case where niceness/priority values should be different, the niceness values (in the case of SCHED_OTHER) of the children processes were set based on the current iterator value using the set_priority function while priority values (in the case of SCHED_RR and SCHED_FIFO) were set based on the current iterator value using the set_scheduler function. I was able to confirm that niceness values were being set in the case of SCHED_OTHER using the get_priority function. In contrast, I was not able to reliably confirm that SCHED_RR and SCHED_FIFO's priority levels were being set because I was not able to find a method to determine priority for real time scheduling. However, based on the fact that the set_scheduler function takes as input a priority level ranging from one to ninety-nine and the fact that I noted different results for tests utilizing different priority levels would indicate that I was successful. Additionally, utilizing the sched_rr_get_interval function allowed me to determine that the time slice quantum did not change after altering the priorities of individual processes in the case of SCHED_RR.

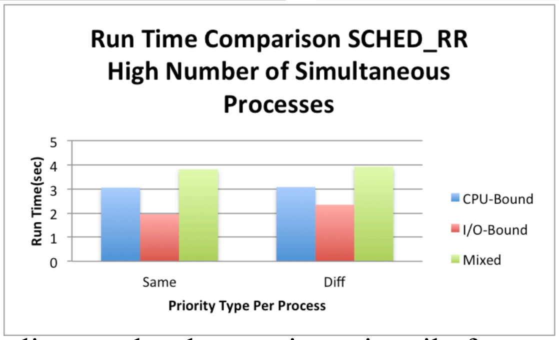
In order to collect the relevant data to answer the questions posed by this assignment, a bash testscript similar to the testscript provided by Sayler and Mishra (2016) was created in order to execute the fifty-four test vectors running under the time command in Unix. Four trials of this testscript were performed and the output data representing the execution time, CPU-time in kernel mode, CPU-time in user mode, percentage use of the CPU, and counts of involuntary and voluntary context switches were placed into appropriately named output files. The collected data was then averaged over the four trials, averaged once more over the number of spawned processes so that values were represented on a per process basis, and synthesized into a PA4Results.xls file for further analysis. Graphical representations of relevant findings may be found in the following section and all synthesized data (PA4Results.xls) may be found in Appendix A. All programs were run using the Fall 2016 Edition of The University of Colorado Computer Science Virtual Machine with Ubuntu version 4.4.16.

Results

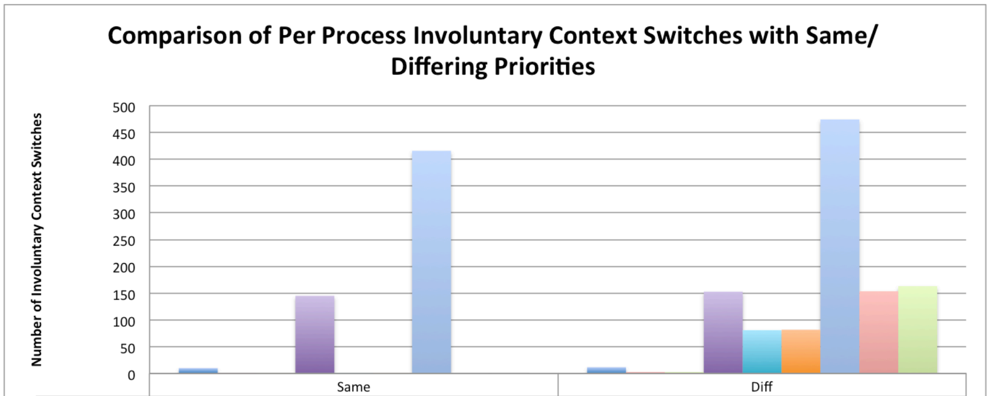
Based on the data collected and synthesized in PA4Results.xls, it was determined that the number of simultaneous processes did not greatly alter the **per-process** run times, number of involuntary context switches, number of voluntary context switches and CPU seconds spent in user or kernel mode using any of the policies. Additionally, individual process run times on all scheduling policies showed that I/O-bound processes ran faster than CPU-bound processes with mixed processes being the slowest of the three process types. This finding was especially profound at the HIGH level of simultaneous processes and can be visualized graphically below showing that I/O-bound processes scale much better than CPU-bound or mixed processes regardless scheduling policy or differing or similar process priorities. It may also be noted below that the SCHED_OTHER (CFS) policy appears to scale better than SCHED_FIFO and SCHED_RR in terms of overall run time across the three process types.



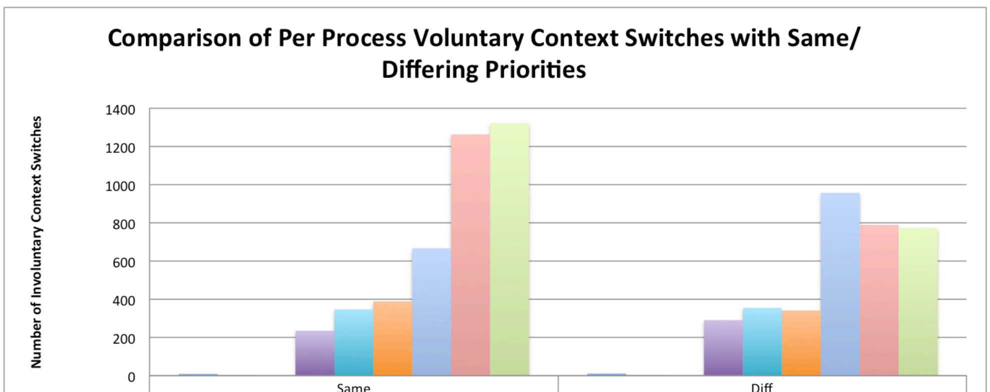




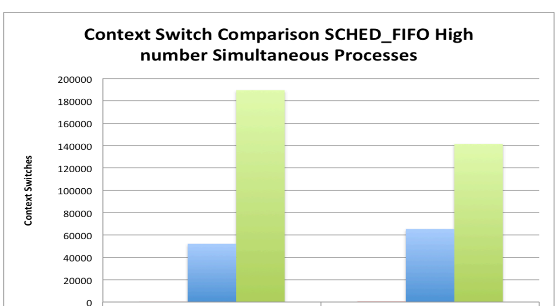
In examining policy overhead stemming primarily from context switches, it was determined that utilizing differing niceness/priorities caused a dramatic increase in involuntary context switches while using SCHED_FIFO and SCHED_RR across all process types with SCHED_OTHER showing a very small to no increase in involuntary context switches. This makes sense because SCHED_FIFO and SCHED_RR are preemptive scheduling policies. In the case of voluntary context switches, utilizing differing niceness/priorities caused a small increase in voluntary context switches with SCHED_OTHER, SCHED_FIFO, and SCHED_RR with I/O bound, and CPU-bound and mixed processes using SCHED_OTHER. However it was also noted that there was a decrease in voluntary context switches for mixed processes utilizing SCHED_FIFO and SCHED_RR. In terms of total context switches, which correspond to total overhead efficiency, using mixed processes with differing niceness/priorities generally have fewer total context switches on SCHED_FIFO and SCHED_RR when compared to utilizing processes with the same niceness/priorities. In contrast, utilizing differing/niceness priorities with mixed processes under SCHED_OTHER results in more total context switches. The graphs below show the differences between the different process types in terms of overhead scaling under the different policies.



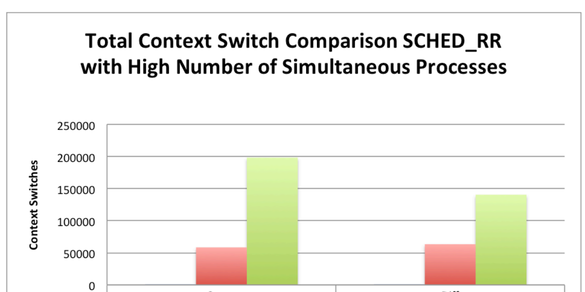
	Same	Diff
CPU_OTHER	9.856	11.5166
CPU_FIFO	0.07	2.4566
CPU_RR	0.061	2.4455
IO_OTHER	144.8983	152.62
IO_FIFO	0.0316	80.9266
IO_RR	0.02666	81.651
MIXED_OTHER	415.7	474.326
MIXED_FIFO	0.01	153.5416
MIXED_RR	0.03	163.0016



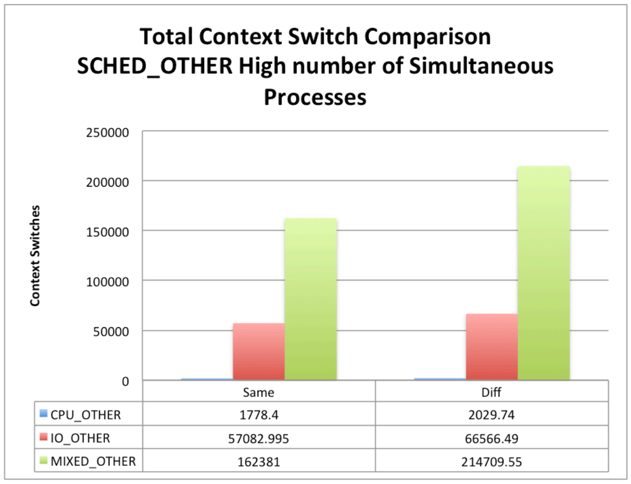
	Same	Diff
CPU_OTHER	9.856	11.5166
CPU_FIFO	1.0466	2.025
CPU_RR	1.045	2.0233
IO_OTHER	235.655	291.1566
IO_FIFO	347.621	355.06
IO_RR	389.553	340.938
MIXED_OTHER	666.84	957.071
MIXED_FIFO	1263.648	790.04
MIXED_RR	1321.8	773.428



	1	2
CPU_FIFO	167.49	672.24
IO_FIFO	52147.89	65397.99
MIXED_FIFO	189548.7	141537.24



	Same	Diff
CPU_RR	165.9	670.32
IO_RR	58436.949	63388.35
MIXED_RR	198274.5	140464.44



Ultimately, after analyzing the data it may be discerned that mixed processes will have the highest number of context switches, I/O-bound processes will have the second most context switches, and CPU-bound processes will have the fewest number of context switches which is to be expected considering the specific goals of each of the different process types. Additionally, the type of scheduling policy and the distinction to use the same versus differing priorities will affect the total number of context switches and the run times. SCHED_OTHER generally results in the highest number of context switches but the lowest run times, SCHED_FIFO generally results in the fewest context switches but the highest run times, and SCHED_RR exists somewhere in the middle of this spectrum. These results will be further discussed in the Analysis section where the answers to the questions posed in the Introduction will be presented.

Analysis

In order to address the questions posed in the Introduction, it is first necessary to explore the pros and cons of each scheduling policy. Based on the data collected from the fifty-four test vectors, in terms overhead efficiency for increasing numbers of simultaneous processes, SCHED_FIFO and SCHED_RR perform similarly in terms of total context switches with SCHED_OTHER generally resulting in the highest number of context switches. However, it should be noted that SCHED_FIFO's total context switches are primarily voluntary while SCHED_RR's total context switches are primarily involuntary. Additionally, making the distinction that each simultaneous process needs to have a different priority causes SCHED_FIFO's total number of involuntary context switches to increase. In terms of run times for increasing numbers of simultaneous processes, SCHED_OTHER resulted in the fastest run times, followed by SCHED_RR and SCHED_FIFO, which perform quite similarly. It is important to note, however, that Linux does not guarantee that it can adhere to the rules outlined by real time scheduling policies such as SCHED_RR and SCHED_FIFO but tries its best. This may muddle some of the data collected from this experiment and explain similarities found between the two real time scheduling policies.

Based on these findings it may be expected that the best scheduling policy for a CPU-bound process to minimize run time is SCHED_OTHER while SCHED_FIFO and SCHED_RR perform quite similarly as the best policy for minimizing total context switch overhead. These findings were consistent regardless of whether individual processes had the same or differing niceness/priorities. Similarly to CPU-bound processes, the best scheduling policy for an I/O bound process to minimize run time is SCHED_OTHER while SCHED_FIFO was the best policy to minimize total context switch overhead. These findings were also consistent when individual processes had the same or differing niceness/priorities with SCHED_RR performing slightly better than SCHED_FIFO in minimizing context switch overhead. Finally, based on the data it was determined that for mixed processes the best scheduling policy to minimize run times was again SCHED_OTHER regardless of niceness/priority for simultaneous processes. However, the best scheduling policy to minimize context switch overhead when niceness values were the same was SCHED_OTHER while SCHED_RR and SCHED_FIFO again performed similarly in terms of minimizing context switch overhead when niceness values were different for simultaneous processes.

Based on the findings from the fifty-four test vectors it is possible to determine what the best scheduling policy would be depending on the type of task that needs to be scheduled based on the current goals of the system, but only after certain tradeoffs are considered. If the operating system was designed in order to minimize run-time on all tasks, then I/O-bound, CPU-bound, and mixed processes will be scheduled under CFS scheduling with SCHED_OTHER. It is easy to imagine that most operating systems would have the primary goal of minimizing run time in order to help users accomplish as many tasks as completely as possible. However, should overhead efficiency be a critical concern for the operating system it is not as clear on which policy would be most beneficial to the task in order to minimize context switch overhead. Definite tradeoffs exist in operating system design and it is up to the developers to decide what aspects of performance are most important. SCHED_OTHER is likely not the best policy to utilize when dealing with singularly CPU-bound or IO-bound tasks, but deals well when trying to optimize run times, making it a useful policy when dealing with mixed processes. In contrast, SCHED_FIFO is an excellent choice when trying to obtain optimal overhead efficiency in terms of tasks that will mainly be performing voluntary context switches as in I/O-bound tasks and CPU-bound tasks. Similarly, SCHED_RR performs quite similarly to SCHED_FIFO and would be well suited to tasks that perform voluntary context switches as in I/O tasks. However, neither SCHED_RR nor SCHED_FIFO are able to minimize run times as well as SCHED_OTHER, which is a definite tradeoff consideration.

Conclusion

Analysis of the data presented in this study would indicate that CFS scheduling with SCHED_OTHER is the most sensible choice to have as the default scheduling policy for an operating system. This is because most processes that the operating system will be scheduling will be mixed processes, with differing priorities, and the operating system will not be able to determine the process type will be prior to running. For example, even if an I/O bound program would benefit well from SCHED_FIFO, the operating system does not know the process is I/O-bound and therefore cannot plan and allocate a scheduling policy ahead of time. Therefore, in spite of the fact that `SCHED_OTHER` results in the highest number of context switches, the fact that it reduces run times across all process types at high levels of simultaneous processes would indicate that the tradeoff between overhead efficiency and run time would be weighted towards minimizing run time during operating system design. However, it is still important to have multiple different kinds of scheduling policies so that the most optimal scheduling policy will be chosen once the activity of the running process is known.

References

[1] Linux Documentation. (n.d.). Retrieved November 05, 2016, from https://linux.die.net/

[2] Sayler, A., Mishra, S., (2016). pi-sched.c [Computer program]. University of Colorado at Boulder.

[3] Sayler, A., & Mishra, S., (2016). rw.c [Computer program]. University of Colorado at Boulder.

[4] Sayler, A., & Mishra, S., (2016). testscript [Computer program]. University of Colorado at Boulder.

[5] Silberschatz, A., Galvin, P. B., & Gagne, G. (2013). *Operating system concepts* (9th ed.). Hoboken, NJ: J. Wiley & Sons.

Appendix A

*All data represented below may also be viewed in PA4Results.xls

*To view raw collected data please review files stored in `/Results` folder

KEY	**Measurement**
wall (%e)	elapsed real time (sec)
user (%U)	CPU-seconds spent in user mode
system (%S)	CPU-seconds spent in kernel mode
CPU (%P = (%S+%U)/%E)	percentage of CPU job used
i-switched (%c)	involuntary context switches; ie expired timeslice
v-switched (%w)	voluntary context switch; ie waiting for i/o

*** LOW = 10, MEDIUM = 50, HIGH = 150 processes**

CPU_SAME_OTHER	**LOW**	**MEDIUM**	**HIGH**	**CPU_DIFF_OTHER**	**LOW**	**MEDIUM**	**HIGH**
wall	0.2025	0.9375	2.835	wall	0.205	0.965	2.9475
i-switched	98	517.25	1478.4	i-switched	175.5	565.75	1727.49
v-switched	20.25	100	300	v-switched	32	104.25	302.25
wall (per process)	0.02025	0.01875	0.0189	wall (per process)	0.0205	0.0193	0.01965
user (per process)	0.036	0.03585	0.03578	user (per process)	0.0365	0.03595	0.0372
system (per process)	0.00025	0.00005	0.003	system (per process)	0	0.0002	0.00028
CPU (per process)	17.75	3.815	1.27	CPU (per process)	17.55	3.735	1.266
i-switched (per process)	9.8	10.345	9.856	i-switched (per process)	10.75	11.315	11.5166
v-switched (per process)	2.025	2	2	v-switched (per process)	3.2	2.085	2.015
total context switches	118.25	617.25	1778.4	total context switches	207.5	670	2029.74
CPU_SAME_FIFO	**LOW**	**MEDIUM**	**HIGH**	**CPU_DIFF_FIFO**	**LOW**	**MEDIUM**	**HIGH**

wall	0.2075	0.9725	3.03	wall	0.2	0.99	3.018
i-switched	5	5.625	10.5	i-switched	28.5	123	368.49
v-switched	15.75	56.75	156.99	v-switched	23.5	103.25	303.75
wall (per process)	0.02075	0.01945	0.0202	wall (per process)	0.02	0.0198	0.02012
user (per process)	0.03875	0.037	0.0368	user (per process)	0.03575	0.03625	0.0365
system (per process)	0	0	0.000083	system (per process)	0	0.0003	0.00023
CPU (per process)	18.6	3.79	1.215	CPU (per process)	17.85	3.69	1.215
i-switched (per process)	0.5	0.1125	0.07	i-switched (per process)	2.85	2.46	2.4566
v-switched (per process)	1.575	1.135	1.0466	v-switched (per process)	2.35	2.065	2.025
total context switches	20.75	62.375	167.49	total context switches	52	226.25	672.24
CPU_SAME_RR	**LOW**	**MEDIUM**	**HIGH**	**CPU_DIFF_RR**	**LOW**	**MEDIUM**	**HIGH**
wall	0.225	1.0375	3.0525	wall	0.205	1.0025	3.075
i-switched	4	5.9375	9.15	i-switched	26.75	125.25	366.825
v-switched	16	56	156.75	v-switched	23.5	105	303.495
wall (per process)	0.0225	0.02075	0.02035	wall (per process)	0.0205	0.02005	0.0205
user (per process)	0.037	0.03775	0.03706	user (per process)	0.03625	0.03685	0.03706
system (per process)	0	0	0.00003	system (per process)	0	0.003	0.00025
CPU (per process)	16.5	3.635	1.21	CPU (per process)	17.65	3.68	1.21
i-switched (per process)	0.4	0.11875	0.061	i-switched (per process)	2.675	2.505	2.4455
v-switched (per process)	1.6	1.12	1.045	v-switched (per process)	2.35	2.1	2.0233
total context switches	20	61.9375	165.9	total context switches	50.25	230.25	670.32
IO_SAME_OTHER	**LOW**	**MEDIUM**	**HIGH**	**IO_DIFF_OTHER**	**LOW**	**MEDIUM**	**HIGH**
wall	0.16	0.52	1.47	wall	0.165	0.5475	1.53
i-switched	1201.75	23755	21734.745	i-switched	1448.25	6274.25	22893
v-switched	2346.75	29645.625	35348.25	v-switched	2569	11820.5	43673.49
wall (per process)	0.016	0.0104	0.0098	wall (per process)	0.0165	0.01095	0.0102
user (per process)	0	0.00005	0.00015	user (per process)	0	0	0.000183
system (per process)	0.0105	0.0098	0.01015	system (per process)	0.011	0.0093	0.0096
CPU (per process)	6.925	1.91	0.6966	CPU (per process)	6.75	1.715	0.6433
i-switched (per process)	120.175	475.1	144.8983	i-switched (per process)	144.825	125.485	152.62
v-switched (per process)	234.675	592.9125	235.655	v-switched (per process)	256.9	236.41	291.1566
total context switches	3548.5	53400.625	57082.995	total context switches	4017.25	18094.75	66566.49
IO_SAME_FIFO	**LOW**	**MEDIUM**	**HIGH**	**IO_DIFF_FIFO**	**LOW**	**MEDIUM**	**HIGH**
wall	0.1675	0.695	2.175	wall	0.1775	0.76	2.6325
i-switched	6	4.25	4.74	i-switched	908.25	4407.25	12138.99
v-switched	2900.5	16483.75	52143.15	v-switched	2797.25	17021	53259
wall (per process)	0.01675	0.0139	0.0145	wall (per process)	0.01775	0.0152	0.01755
user (per process)	0	0.00015	0.0001	user (per process)	0	0.00015	0.00025
system (per process)	0.01375	0.01455	0.01583	system (per process)	0.01725	0.0179	0.02011
CPU (per process)	8.225	2.11	0.72833	CPU (per process)	9.8	2.38	0.7716
i-switched (per process)	0.6	0.085	0.0316	i-switched (per process)	90.825	88.145	80.9266
v-switched (per process)	290.05	329.675	347.621	v-switched (per process)	279.725	340.42	355.06
total context switches	2906.5	16488	52147.89	total context switches	3705.5	21428.25	65397.99
IO_SAME_RR	**LOW**	**MEDIUM**	**HIGH**	**IO_DIFF RR**	**LOW**	**MEDIUM**	**HIGH**
wall	0.175	0.6625	1.9575	wall	0.175	0.895	2.3445
i-switched	2.75	3.5	3.999	i-switched	927	4324.5	12247.65
v-switched	2766.25	16263.5	58432.95	v-switched	2810.5	17040.5	51140.7
wall (per process)	0.0175	0.01325	0.01305	wall (per process)	0.0175	0.0179	0.01563
user (per process)	0	0.00005	0.000183	user (per process)	0	0.0001	0.000216
system (per process)	0.015	0.0144	0.01466	system (per process)	0.01725	0.0066	0.01868

CPU (per process)	8.5	2.185	0.7566	CPU (per process)	10.075	2.235	0.80333
i-switched (per process)	0.275	0.07	0.02666	i-switched (per process)	92.7	86.49	81.651
v-switched (per process)	276.625	325.27	389.553	v-switched (per process)	281.05	340.81	340.938
total context switches	2769	16267	58436.949	total context switches	3737.5	21365	63388.35
mix_SAME_OTHER	**LOW**	**MEDIUM**	**HIGH**	**mix_DIFF_OTHER**	**LOW**	**MEDIUM**	**HIGH**
wall	0.3575	1.49	3.42	wall	0.375	1.25	3.6495
i-switched	2750.25	15626.75	62355	i-switched	6292.75	23921.5	71148.9
v-switched	3900.75	16677	100026	v-switched	4910.5	27807.5	143560.65
wall (per process)	0.03575	0.0298	0.0228	wall (per process)	0.0375	0.025	0.02433
user (per process)	0.00475	0.0043	0.00435	user (per process)	0.00375	0.0041	0.0048
system (per process)	0.016	0.0135	0.01936	system (per process)	0.027	0.02035	0.0207
CPU (per process)	6	1.255	0.6933	CPU (per process)	8.3	1.94	0.695
i-switched (per process)	275.025	312.535	415.7	i-switched (per process)	629.275	478.43	474.326
v-switched (per process)	390.075	333.54	666.84	v-switched (per process)	491.05	556.15	957.071
total context switches	6651	32303.75	162381	total context switches	11203.25	51729	214709.55
mix_SAME_FIFO	**LOW**	**MEDIUM**	**HIGH**	**mix_DIFF_FIFO**	**LOW**	**MEDIUM**	**HIGH**
wall	0.3275	1.4425	3.81	wall	0.3575	1.335	4.194
i-switched	3.75	3.25	1.5	i-switched	2283.75	9596.25	23031.24
v-switched	6233	43400.75	189547.2	v-switched	5672.25	31548	118506
wall (per process)	0.03275	0.02885	0.0254	wall (per process)	0.03575	0.0267	0.02796
user (per process)	0.00425	0.0061	0.00608	user (per process)	0.00475	0.0058	0.0057
system (per process)	0.02925	0.0301	0.026116	system (per process)	0.03675	0.0315	0.03278
CPU (per process)	10.275	2.495	0.8433	CPU (per process)	11.75	2.79	0.9466
i-switched (per process)	0.375	0.065	0.01	i-switched (per process)	228.375	191.925	153.5416
v-switched (per process)	623.3	868.015	1263.648	v-switched (per process)	567.225	630.96	790.04
total context switches	6236.75	43404	189548.7	total context switches	7956	41144.25	141537.24
mix_SAME_RR	**LOW**	**MEDIUM**	**HIGH**	**mix_DIFF_RR**	**LOW**	**MEDIUM**	**HIGH**
wall	0.3025	1.185	3.8145	wall	0.335	1.355	3.9225
i-switched	2.25	2.75	4.5	i-switched	2260	9224.75	24450.24
v-switched	6542.75	44331.75	198270	v-switched	5539.5	31512.25	116014.2
wall (per process)	0.03025	0.0237	0.02543	wall (per process)	0.0335	0.0271	0.02615
user (per process)	0.00575	0.0047	0.00581	user (per process)	0.0045	0.00595	0.00585
system (per process)	0.02525	0.0256	0.02674	system (per process)	0.03625	0.0323	0.03115
CPU (per process)	10.325	2.55	0.8816	CPU (per process)	12.175	2.81	0.9433
i-switched (per process)	0.225	0.055	0.03	i-switched (per process)	226	184.495	163.0016
v-switched (per process)	654.275	886.635	1321.8	v-switched (per process)	553.95	630.245	773.428

Appendix B

* All files located within submission file.

1. **cpu-bound.c** A simple program CPU-bound program for statistically calculating the value of pi using a specified number of simultaneous processes, specified scheduling policy, and optional scheduling niceness/priority distinction. Adapted from pi-sched.c by Sayler and Mishra (2016).

2. **io-bound.c** A small I/O bound program to copy N bytes from an input file to an output file using a specified number of simultaneous processes, specified scheduling policy, and

optional scheduling niceness/priority distinction. May read the input file multiple times if N is larger than the size of the input file. Adapted from rw.c by Sayler and Mishra (2016).

3. **mixed.c** A mixture of cpu-bound.c and io-bound.c which computes the value of pi over one hundred thousand iterations, performing an I/O operation every one thousandth iteration. This program is a representation of a mixed process.

4. **testscript2** A simple bash script to run fifty-four test vectors and output results to appropriately named files stored within the /data folder.

5. **Makefile** A GNU Make makefile to build all code listed above.

6. **README** A description of the submission contents and how to build and run all code and test vectors.

7. **/Results** A folder which contains PA4Results.xls and /PA4Graphs as seen in the above report.

8. **/data** A folder containing the output of the results obtained by running testscript2.