Andrew Candelaresi

Linux Scheduler Analysis

Abstract:

For this paper we looked at three different Linux schedulers. These were the CFS scheduler, (used most by Linux OS) First in First Out Scheduler fifo, and Round Robin Scheduler RR. To asses the ability of each of these schedulers we ran them each with three different types of processes, one that was computation bound, one that was I/O bound and one that was a mix of both. Then we ran each of these three different processes at three different sizes a small, medium and large versions. We took the results of this process and analyzed the data looking at the amount of user time, cpu time, and the number of voluntary and involuntary context switches. Based on these data points we developed some conclusions about which schedulers performed best under which conditions.

INTRODUCTION:

Coming up with efficient process scheduling policy is a task that computer engineers have been working on for years. Many different strategies have been developed over the past decades. A few of these scheduling policies have been shown through proof and analysis to be significantly better than others. For this project we have taken three such processes and and analyzed their run time with three different types of processes using a large, medium and small amounts of simultaneous processes. This way we can see not only how these Schedulers compare with different types of processes, but also how the algorithms used in each scheduler scale as the size of data .

METHOD:

The different types of processes we used are co, I/O bound, and a mix of computation and I/O bound. We used a program called pi which uses a randomized process that generates the number pi. This program is compute bound meaning that it uses up lots of CPU doing calculations. The program generates a more accurate number the higher the iterations are, so we used 100,000 iterations. This was our first developed benchmark.

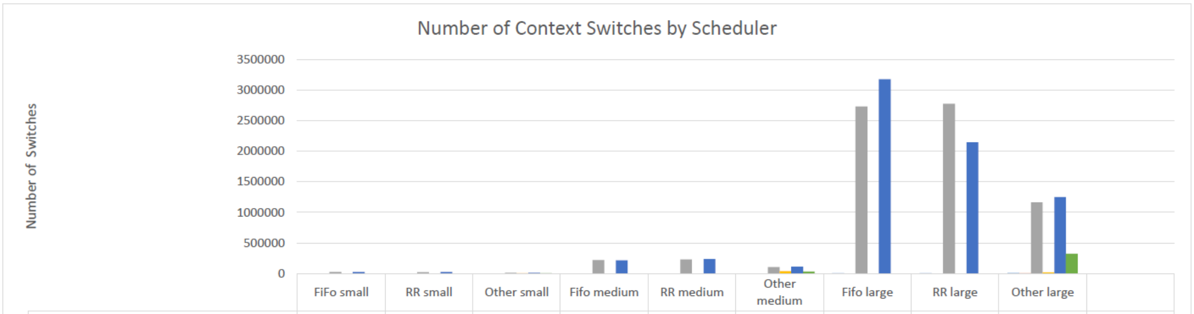
We used a program that copies N bytes form a input file to an output file. The process writes 102400 bytes in blocks of 1024.

For the mixed I/O and compute bound we combined the pi process with the write to disk process, using the same parameters of iterations and writes as a benchmark.

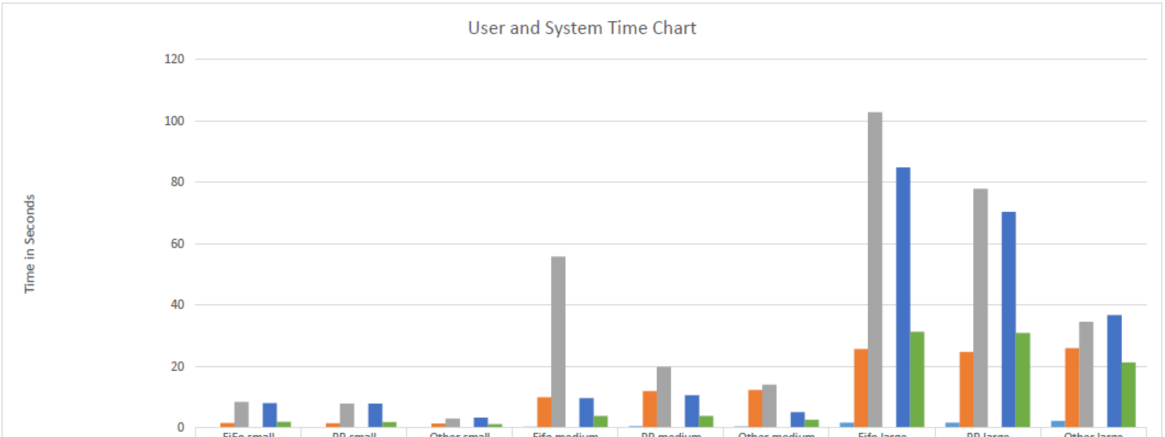
We then developed size benchmarks as follows. The number of processes running simultaneously was considered to be small at 64, medium at 512 and large at 4096.

A bash script was written that would run all the test process in succession. We used a makefile to compile all off the test process. We set the -wall and -wesxtra error checking flags set to check for any errors. This bash script ran the pi process on each scheduler with the smallest simultaneous processes. Then it ran the the I/O bound processes with the smallest simultaneous processes. it then proceeded to run the mixed I/O process with the smallest simultaneous processes. Once all the processes had finished running with the smallest number of simultaneous processes it the script runs the medium (512) simultaneous processes in for each process and scheduler in the same order. Finally the largest (4096) simultaneous processes are run on all schedulers with all processes.

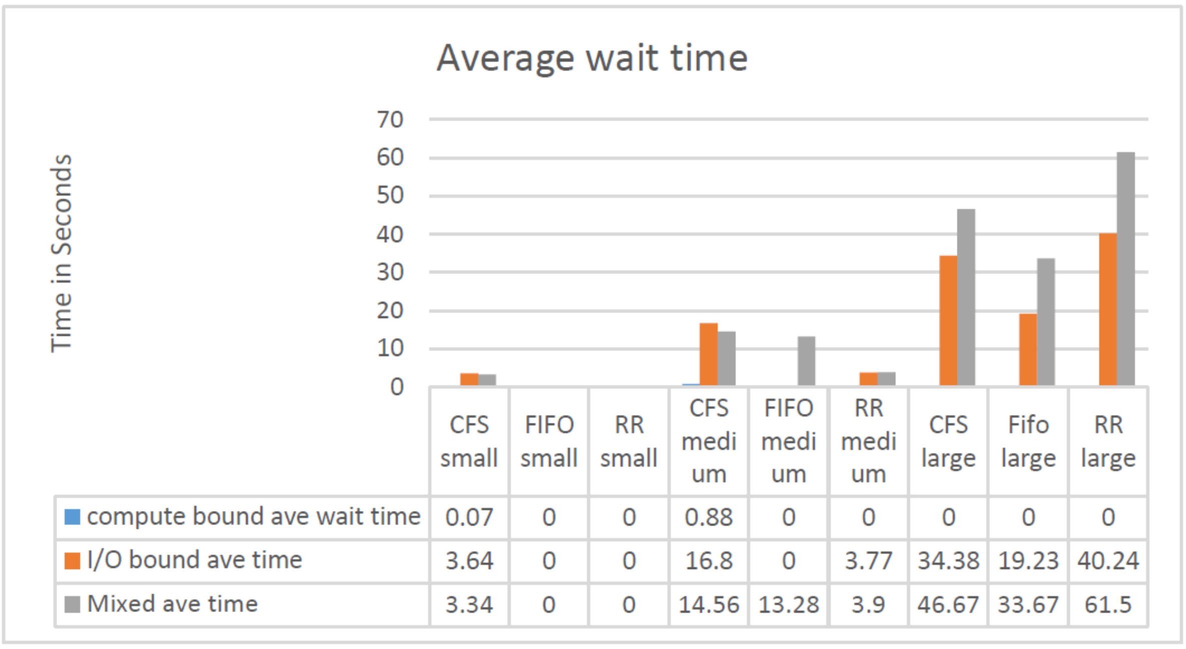
RESULTS:
We have provided graphs which show a clear side by side comparison of the important factors. In the first graph these factors are the amount of user time and system time dedicated to each process by each scheduler across the small medium and large sets. In the second graph we show a comparison of involuntary and voluntary context switches for each process across all sets.



	FiFo small	RR small	Other small	Fifo medium	RR medium	Other medium	Fifo large	RR large	Other large
■ compute bound volunatry context switches	130	130	130	1026	1026	1257	8191	8177	10424
■ compute bound involuntary context switches	7	21	434	13	15	4150	29	24	7607
■ I/O bound volunatry context switches	27220	26321	12829	219599	230746	106307	2726746	2775705	1162827
■ I/O bound involuntary context switches	11	22	7399	21	27	38396	39	19	20383
■ Mixed volunatry context switches	27092	27383	12659	217268	237142	114575	3172366	2147220	1248965
■ Mixed involuntary context switches	7	26	8573	12	6	33890	22	49	323636



	FiFo small	RR small	Other small	Fifo medium	RR medium	Other medium	Fifo large	RR large	Other large
■ compute bound system time	0.08	0.06	0.24	0.43	0.62	0.53	1.75	1.81	2.31
■ compute bound user time	1.69	1.6	1.43	10.05	12.04	12.38	25.74	24.81	26.02
■ I/O bound system time	8.5	7.94	3.09	55.77	19.97	14.16	102.78	77.87	34.61
■ I/O bound user time	0.07	0.05	0.01	0.31	0.19	0.11	0.3	0.22	0.14
■ Mixed system time	8.17	7.93	3.32	9.74	10.64	5.2	84.8	70.28	36.74
■ Mixed User time	2.06	1.93	1.26	3.94	3.99	2.71	31.31	30.94	21.32



	CFS small	FIFO small	RR small	CFS medium	FIFO medium	RR medium	CFS large	Fifo large	RR large
🟦 compute bound ave wait time	0.07	0	0	0.88	0	0	0	0	0
🟧 I/O bound ave time	3.64	0	0	16.8	0	3.77	34.38	19.23	40.24
⬜ Mixed ave time	3.34	0	0	14.56	13.28	3.9	46.67	33.67	61.5

ANALYSIS:
We were expecting to see that the CFS scheduler performs the best since it is the Scheduler that is currently used by the Linux OS and stands to reason that they would implement the best scheduling policy.

From the results we can see important patterns that can help decide which scheduling algorithms perform best in each situation. A few general analysis first, as the number of simultaneous process being run increased the time dedicated to user time in compute bound increased, the amount of system time in I/O bound increased and the amount of both system and user time increased for the mixed compute and I/O bound process. This was a result that we were expecting and it was shown to be accurate. So we can determine that compute bound processes spend most of their time in the user space, while I/O bound processes spend most of their time in the kernel space.

With context switching we expected that the RR would have the most voluntary context switches followed by CFS and FIFO. For involuntary context switches we expected the FIFO to have the most followed by the CFS and the RR. Our prediction was inaccurate in both cases. The CFS has the most involuntary context switches for all size samples. Looking at the I/O bound context switches the RR and FIFO had very low involuntary context switches compared to the amount of voluntary context switches. For I/O bound medium size sample the RR and FIFO have 99.99% voluntary context switches. For the same process and sample size the CFS scheduler had 31% involuntary context switches. These finding surprised us since we expected to see higher involuntary context switches from the FIFO and lower involuntary context switches from the CFS. Also as the amount of simultaneous processes increased the number of voluntary and involuntary context switches increased. From this we can see that the more processes that are running in parallel the more context switches we can expect to see the CPU to make.

From this data we can start to understand which scheduler would be best suited for each different type of process and if this holds for all levels of concurrency. Looking at time the Other scheduler (Linux_CFS) has the best performance by a very slight amount through all sizes of simultaneous runs. The difference between the three schedulers at the medium size is smaller than at any other set size I would imagine this is do to the amount of involuntary context switching that occurs with the Other scheduler. Considering I/O bound processes, again we see the CFS scheduler consuming the least amount of time. They are all close to the same amount of time in the small size, the FIFO jumps up significantly at the medium size while the RR stays relatively close to the CFS. At the large size the FIFO is triple the time of the CFS and the RR is double the CFS. This significant gap as the size of the concurrent processes shows us that clearly the CFS is best suited for I/O bound processes. Looking at the mixed I/O and compute bound process again we see a very large jump in the system time for all three processes, 70 for FIFO, 60 for RR, and 30 for CFS.

CONCLUSION

It is quite clear from the data provided that in terms of time the CFS scheduler out performs both the FIFO and the RR. We can develop a list of pro and cons from the data gathered in this report. CFS, pros: efficient run time allocation, this scheduler has the lowest time spent in system and user time of all the schedulers. Cons: high context switching and longer wait times. FIFO Pros: lower wait time, lower involuntary context switches. Cons: much higher system times necessary to complete I/O bound and mixed processes. RR, Pros: lowest wait times due to higher voluntary context switches. Cons: larger amount of time spent on system time in I/O bound processes. The use of CFS would be recommended for every day use of a computer. The RR scheduler would be good in a situation where you wanted to run a very large number of processes. The FIFO schedule would be good in a situation where you had a small number of Compute bound processes.

APPENDIX A RAW DATA:

```
Building code...
1024+0 records in
1024+0 records out
1048576 bytes (1.0 MB) copied, 0.310337 s, 3.4 MB/s
Starting test runs...
Calculating pi over 100000 iterations using SCHED_OTHER with 64 simultaneous
processes...
        Command being timed: "sudo ./pi-sched 100000 SCHED_OTHER 6"
        User time (seconds): 1.43
        System time (seconds): 0.24
        Percent of CPU this job got: 96%
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:01.74
        Average shared text size (kbytes): 0
        Average unshared data size (kbytes): 0
        Average stack size (kbytes): 0
        Average total size (kbytes): 0
        Maximum resident set size (kbytes): 4288
        Average resident set size (kbytes): 0
        Major (requiring I/O) page faults: 0
        Minor (reclaiming a frame) page faults: 3022
```

```
	Voluntary context switches: 130
	Involuntary context switches: 434
	Swaps: 0
	File system inputs: 0
	File system outputs: 520
	Socket messages sent: 0
	Socket messages received: 0
	Signals delivered: 0
	Page size (bytes): 4096
	Exit status: 0
Calculating pi over 100000 iterations using SCHED_FIFO with 64 simultaneous processes...
	Command being timed: "sudo ./pi-sched 100000 SCHED_FIFO 6"
	User time (seconds): 1.69
	System time (seconds): 0.08
	Percent of CPU this job got: 191%
	Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.93
	Average shared text size (kbytes): 0
	Average unshared data size (kbytes): 0
	Average stack size (kbytes): 0
	Average total size (kbytes): 0
	Maximum resident set size (kbytes): 4300
	Average resident set size (kbytes): 0
	Major (requiring I/O) page faults: 0
	Minor (reclaiming a frame) page faults: 3021
	Voluntary context switches: 130
	Involuntary context switches: 7
	Swaps: 0
	File system inputs: 0
	File system outputs: 528
	Socket messages sent: 0
	Socket messages received: 0
	Signals delivered: 0
	Page size (bytes): 4096
	Exit status: 0
Calculating pi over 100000 iterations using SCHED_RR with 64 simultaneous processes...
	Command being timed: "sudo ./pi-sched 100000 SCHED_RR 6"
	User time (seconds): 1.60
	System time (seconds): 0.06
	Percent of CPU this job got: 181%
	Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.92
	Average shared text size (kbytes): 0
	Average unshared data size (kbytes): 0
	Average stack size (kbytes): 0
	Average total size (kbytes): 0
	Maximum resident set size (kbytes): 4192
	Average resident set size (kbytes): 0
	Major (requiring I/O) page faults: 0
	Minor (reclaiming a frame) page faults: 3081
	Voluntary context switches: 130
	Involuntary context switches: 21
	Swaps: 0
	File system inputs: 0
	File system outputs: 520
	Socket messages sent: 0
	Socket messages received: 0
	Signals delivered: 0
	Page size (bytes): 4096
	Exit status: 0
Copying 102400 bytes in blocks of 1024 from rwinput to rwoutput
using SCHED_OTHER with 64 simultaneous processes...
	Command being timed: "sudo ./rw 102400 1024 rwinput rwoutput 6
SCHED_OTHER"
	User time (seconds): 0.01
	System time (seconds): 3.09
	Percent of CPU this job got: 46%
	Elapsed (wall clock) time (h:mm:ss or m:ss): 0:06.74
	Average shared text size (kbytes): 0
	Average unshared data size (kbytes): 0
	Average stack size (kbytes): 0
	Average total size (kbytes): 0
	Maximum resident set size (kbytes): 4240
	Average resident set size (kbytes): 0
	Major (requiring I/O) page faults: 0
	Minor (reclaiming a frame) page faults: 2795
	Voluntary context switches: 12829
	Involuntary context switches: 7399
	Swaps: 0
	File system inputs: 0
	File system outputs: 207656
	Socket messages sent: 0
	Socket messages received: 0
	Signals delivered: 0
	Page size (bytes): 4096
	Exit status: 0
Copying 102400 bytes in blocks of 1024 from rwinput to rwoutput
using SCHED_FIFO with 64 simultaneous processes...
	Command being timed: "sudo ./rw 102400 1024 rwinput rwoutput 6 SCHED_FIFO"
	User time (seconds): 0.07
	System time (seconds): 8.50
	Percent of CPU this job got: 104%
	Elapsed (wall clock) time (h:mm:ss or m:ss): 0:08.24
	Average shared text size (kbytes): 0
	Average unshared data size (kbytes): 0
	Average stack size (kbytes): 0
	Average total size (kbytes): 0
	Maximum resident set size (kbytes): 4276
	Average resident set size (kbytes): 0
	Major (requiring I/O) page faults: 0
	Minor (reclaiming a frame) page faults: 2734
	Voluntary context switches: 27220
	Involuntary context switches: 11
	Swaps: 0
	File system inputs: 0
	File system outputs: 393824
	Socket messages sent: 0
	Socket messages received: 0
	Signals delivered: 0
	Page size (bytes): 4096
	Exit status: 0
Copying 102400 bytes in blocks of 1024 from rwinput to rwoutput
using SCHED_RR with 64 simultaneous processes...
	Command being timed: "sudo ./rw 102400 1024 rwinput rwoutput 6 SCHED_RR"
	User time (seconds): 0.05
	System time (seconds): 7.94
	Percent of CPU this job got: 104%
	Elapsed (wall clock) time (h:mm:ss or m:ss): 0:07.65
	Average shared text size (kbytes): 0
	Average unshared data size (kbytes): 0
	Average stack size (kbytes): 0
	Average total size (kbytes): 0
	Maximum resident set size (kbytes): 4268
	Average resident set size (kbytes): 0
	Major (requiring I/O) page faults: 0
	Minor (reclaiming a frame) page faults: 2670
	Voluntary context switches: 26321
	Involuntary context switches: 22
	Swaps: 0
	File system inputs: 0
	File system outputs: 327904
	Socket messages sent: 0
	Socket messages received: 0
	Signals delivered: 0
	Page size (bytes): 4096
	Exit status: 0
Running mixed CPU/IO bound process
using SCHED_OTHER with 64 simultaneous processes...
	Command being timed: "sudo ./mixed 102400 1024 rwinput rwoutput 6
SCHED_OTHER 100000"
	User time (seconds): 1.26
	System time (seconds): 3.32
	Percent of CPU this job got: 57%
	Elapsed (wall clock) time (h:mm:ss or m:ss): 0:07.92
	Average shared text size (kbytes): 0
	Average unshared data size (kbytes): 0
	Average stack size (kbytes): 0
	Average total size (kbytes): 0
	Maximum resident set size (kbytes): 4216
	Average resident set size (kbytes): 0
	Major (requiring I/O) page faults: 0
	Minor (reclaiming a frame) page faults: 3300
	Voluntary context switches: 12659
	Involuntary context switches: 8573
	Swaps: 0
	File system inputs: 0
	File system outputs: 217512
	Socket messages sent: 0
	Socket messages received: 0
	Signals delivered: 0
	Page size (bytes): 4096
	Exit status: 0
Running mixed CPU/IO bound process
using SCHED_FIFO with 64 simultaneous processes...
	Command being timed: "sudo ./mixed 102400 1024 rwinput rwoutput 6
SCHED_FIFO 100000"
	User time (seconds): 2.06
	System time (seconds): 8.17
	Percent of CPU this job got: 115%
	Elapsed (wall clock) time (h:mm:ss or m:ss): 0:08.85
	Average shared text size (kbytes): 0
	Average unshared data size (kbytes): 0
	Average stack size (kbytes): 0
	Average total size (kbytes): 0
	Maximum resident set size (kbytes): 4148
	Average resident set size (kbytes): 0
	Major (requiring I/O) page faults: 0
	Minor (reclaiming a frame) page faults: 3300
	Voluntary context switches: 27092
	Involuntary context switches: 7
	Swaps: 0
	File system inputs: 0
	File system outputs: 414408
	Socket messages sent: 0
	Socket messages received: 0
	Signals delivered: 0
	Page size (bytes): 4096
	Exit status: 0
Running mixed CPU/IO bound process
using SCHED_RR with 64 simultaneous processes...
	Command being timed: "sudo ./mixed 102400 1024 rwinput rwoutput 6 SCHED_RR
100000"
	User time (seconds): 1.93
	System time (seconds): 7.93
	Percent of CPU this job got: 113%
	Elapsed (wall clock) time (h:mm:ss or m:ss): 0:08.69
	Average shared text size (kbytes): 0
	Average unshared data size (kbytes): 0
	Average stack size (kbytes): 0
	Average total size (kbytes): 0
	Maximum resident set size (kbytes): 4216
	Average resident set size (kbytes): 0
	Major (requiring I/O) page faults: 0
	Minor (reclaiming a frame) page faults: 3175
	Voluntary context switches: 27383
	Involuntary context switches: 26
	Swaps: 0
	File system inputs: 0
	File system outputs: 425760
	Socket messages sent: 0
	Socket messages received: 0
	Signals delivered: 0
	Page size (bytes): 4096
	Exit status: 0
Calculating pi over 100000 iterations using SCHED_OTHER with 512 simultaneous
processes...
	Command being timed: "sudo ./pi-sched 100000 SCHED_OTHER 9"
	User time (seconds): 12.38
	System time (seconds): 0.53
	Percent of CPU this job got: 93%
	Elapsed (wall clock) time (h:mm:ss or m:ss): 0:13.79
	Average shared text size (kbytes): 0
	Average unshared data size (kbytes): 0
	Average stack size (kbytes): 0
	Average total size (kbytes): 0
	Maximum resident set size (kbytes): 4276
	Average resident set size (kbytes): 0
	Major (requiring I/O) page faults: 0
	Minor (reclaiming a frame) page faults: 21233
	Voluntary context switches: 1257
	Involuntary context switches: 4150
	Swaps: 0
	File system inputs: 0
	File system outputs: 4200
	Socket messages sent: 0
	Socket messages received: 0
	Signals delivered: 0
	Page size (bytes): 4096
	Exit status: 0
Calculating pi over 100000 iterations using SCHED_FIFO with 512 simultaneous
processes...
	Command being timed: "sudo ./pi-sched 100000 SCHED_FIFO 9"
	User time (seconds): 10.05
	System time (seconds): 0.43
	Percent of CPU this job got: 190%
	Elapsed (wall clock) time (h:mm:ss or m:ss): 0:05.50
	Average shared text size (kbytes): 0
	Average unshared data size (kbytes): 0
	Average stack size (kbytes): 0
	Average total size (kbytes): 0
	Maximum resident set size (kbytes): 4216
	Average resident set size (kbytes): 0
	Major (requiring I/O) page faults: 0
	Minor (reclaiming a frame) page faults: 19691
	Voluntary context switches: 1026
	Involuntary context switches: 13
	Swaps: 0
	File system inputs: 0
	File system outputs: 4184
	Socket messages sent: 0
	Socket messages received: 0
	Signals delivered: 0
	Page size (bytes): 4096
	Exit status: 0
Calculating pi over 100000 iterations using SCHED_RR with 512 simultaneous processes...
	Command being timed: "sudo ./pi-sched 100000 SCHED_RR 9"
	User time (seconds): 12.04
	System time (seconds): 0.62
	Percent of CPU this job got: 190%
	Elapsed (wall clock) time (h:mm:ss or m:ss): 0:06.65
	Average shared text size (kbytes): 0
	Average unshared data size (kbytes): 0
	Average stack size (kbytes): 0
	Average total size (kbytes): 0
	Maximum resident set size (kbytes): 4280
	Average resident set size (kbytes): 0
	Major (requiring I/O) page faults: 0
	Minor (reclaiming a frame) page faults: 21232
	Voluntary context switches: 1026
	Involuntary context switches: 15
	Swaps: 0
	File system inputs: 0
	File system outputs: 4200
	Socket messages sent: 0
	Socket messages received: 0
	Signals delivered: 0
	Page size (bytes): 4096
	Exit status: 0
Copying 102400 bytes in blocks of 1024 from rwinput to rwoutput
using SCHED_OTHER with 512 simultaneous processes...
	Command being timed: "sudo ./rw 102400 1024 rwinput rwoutput 9
SCHED_OTHER"
	User time (seconds): 0.11
	System time (seconds): 14.16
	Percent of CPU this job got: 45%
	Elapsed (wall clock) time (h:mm:ss or m:ss): 0:31.07
	Average shared text size (kbytes): 0
	Average unshared data size (kbytes): 0
	Average stack size (kbytes): 0
	Average total size (kbytes): 0
	Maximum resident set size (kbytes): 4248
	Average resident set size (kbytes): 0
	Major (requiring I/O) page faults: 0
	Minor (reclaiming a frame) page faults: 17389
	Voluntary context switches: 106307
	Involuntary context switches: 38396
	Swaps: 0
	File system inputs: 0
	File system outputs: 857528
	Socket messages sent: 0
	Socket messages received: 0
	Signals delivered: 0
	Page size (bytes): 4096
	Exit status: 0
```

```
Copying 102400 bytes in blocks of 1024 from rwinput to rwoutput
using SCHED_FIFO with 512 simultaneous processes...
	Command being timed: "sudo ./rw 102400 1024 rwinput rwoutput 9 SCHED_FIFO"
	User time (seconds): 0.31
	System time (seconds): 55.77
	Percent of CPU this job got: 125%
	Elapsed (wall clock) time (h:mm:ss or m:ss): 0:44.56
	Average shared text size (kbytes): 0
	Average unshared data size (kbytes): 0
	Average stack size (kbytes): 0
	Average total size (kbytes): 0
	Maximum resident set size (kbytes): 4216
	Average resident set size (kbytes): 0
	Major (requiring I/O) page faults: 0
	Minor (reclaiming a frame) page faults: 18925
	Voluntary context switches: 219599
	Involuntary context switches: 21
	Swaps: 0
	File system inputs: 0
	File system outputs: 2509112
	Socket messages sent: 0
	Socket messages received: 0
	Signals delivered: 0
	Page size (bytes): 4096
	Exit status: 0
Copying 102400 bytes in blocks of 1024 from rwinput to rwoutput
using SCHED_RR with 512 simultaneous processes...
	Command being timed: "sudo ./rw 102400 1024 rwinput rwoutput 9 SCHED_RR"
	User time (seconds): 0.19
	System time (seconds): 19.97
	Percent of CPU this job got: 84%
	Elapsed (wall clock) time (h:mm:ss or m:ss): 0:23.93
	Average shared text size (kbytes): 0
	Average unshared data size (kbytes): 0
	Average stack size (kbytes): 0
	Average total size (kbytes): 0
	Maximum resident set size (kbytes): 4192
	Average resident set size (kbytes): 0
	Major (requiring I/O) page faults: 0
	Minor (reclaiming a frame) page faults: 17385
	Voluntary context switches: 230746
	Involuntary context switches: 27
	Swaps: 0
	File system inputs: 0
	File system outputs: 2112304
	Socket messages sent: 0
	Socket messages received: 0
	Signals delivered: 0
	Page size (bytes): 4096
	Exit status: 0
Running mixed CPU/IO bound process
using SCHED_OTHER with 512 simultaneous processes...
```

```
	Command being timed: "sudo ./mixed 102400 1024 rwinput rwoutput 9 
SCHED_OTHER 100000"
	User time (seconds): 2.71
	System time (seconds): 5.20
	Percent of CPU this job got: 35%
	Elapsed (wall clock) time (h:mm:ss or m:ss): 0:22.47
	Average shared text size (kbytes): 0
	Average unshared data size (kbytes): 0
	Average stack size (kbytes): 0
	Average total size (kbytes): 0
	Maximum resident set size (kbytes): 4216
	Average resident set size (kbytes): 0
	Major (requiring I/O) page faults: 0
	Minor (reclaiming a frame) page faults: 22502
	Voluntary context switches: 114575
	Involuntary context switches: 33890
	Swaps: 0
	File system inputs: 0
	File system outputs: 919744
	Socket messages sent: 0
	Socket messages received: 0
	Signals delivered: 0
	Page size (bytes): 4096
	Exit status: 0
Running mixed CPU/IO bound process
using SCHED_FIFO with 512 simultaneous processes...
	Command being timed: "sudo ./mixed 102400 1024 rwinput rwoutput 9 
SCHED_FIFO 100000"
	User time (seconds): 3.94
	System time (seconds): 9.74
	Percent of CPU this job got: 50%
	Elapsed (wall clock) time (h:mm:ss or m:ss): 0:26.96
	Average shared text size (kbytes): 0
	Average unshared data size (kbytes): 0
	Average stack size (kbytes): 0
	Average total size (kbytes): 0
	Maximum resident set size (kbytes): 4200
	Average resident set size (kbytes): 0
	Major (requiring I/O) page faults: 0
	Minor (reclaiming a frame) page faults: 22514
	Voluntary context switches: 217268
	Involuntary context switches: 12
	Swaps: 0
	File system inputs: 0
	File system outputs: 1350984
	Socket messages sent: 0
	Socket messages received: 0
	Signals delivered: 0
	Page size (bytes): 4096
	Exit status: 0
Running mixed CPU/IO bound process
using SCHED_RR with 512 simultaneous processes...
```

```
	Command being timed: "sudo ./mixed 102400 1024 rwinput rwoutput 9 SCHED_RR
100000"
	User time (seconds): 3.99
	System time (seconds): 10.64
	Percent of CPU this job got: 78%
	Elapsed (wall clock) time (h:mm:ss or m:ss): 0:18.53
	Average shared text size (kbytes): 0
	Average unshared data size (kbytes): 0
	Average stack size (kbytes): 0
	Average total size (kbytes): 0
	Maximum resident set size (kbytes): 4196
	Average resident set size (kbytes): 0
	Major (requiring I/O) page faults: 0
	Minor (reclaiming a frame) page faults: 21485
	Voluntary context switches: 237142
	Involuntary context switches: 6
	Swaps: 0
	File system inputs: 0
	File system outputs: 1602984
	Socket messages sent: 0
	Socket messages received: 0
	Signals delivered: 0
	Page size (bytes): 4096
	Exit status: 0
Calculating pi over 100000 iterations using SCHED_OTHER with 4096 simultaneous
processes...
	Command being timed: "sudo ./pi-sched 100000 SCHED_OTHER 12"
	User time (seconds): 26.02
	System time (seconds): 2.31
	Percent of CPU this job got: 99%
	Elapsed (wall clock) time (h:mm:ss or m:ss): 0:28.55
	Average shared text size (kbytes): 0
	Average unshared data size (kbytes): 0
	Average stack size (kbytes): 0
	Average total size (kbytes): 0
	Maximum resident set size (kbytes): 4196
	Average resident set size (kbytes): 0
	Major (requiring I/O) page faults: 0
	Minor (reclaiming a frame) page faults: 145907
	Voluntary context switches: 10424
	Involuntary context switches: 7607
	Swaps: 0
	File system inputs: 0
	File system outputs: 33536
	Socket messages sent: 0
	Socket messages received: 0
	Signals delivered: 0
	Page size (bytes): 4096
	Exit status: 0
Calculating pi over 100000 iterations using SCHED_FIFO with 4096 simultaneous
processes...
	Command being timed: "sudo ./pi-sched 100000 SCHED_FIFO 12"
	User time (seconds): 25.74
	System time (seconds): 1.75
	Percent of CPU this job got: 190%
	Elapsed (wall clock) time (h:mm:ss or m:ss): 0:14.44
	Average shared text size (kbytes): 0
	Average unshared data size (kbytes): 0
	Average stack size (kbytes): 0
	Average total size (kbytes): 0
	Maximum resident set size (kbytes): 4252
	Average resident set size (kbytes): 0
	Major (requiring I/O) page faults: 0
	Minor (reclaiming a frame) page faults: 162298
	Voluntary context switches: 8191
	Involuntary context switches: 29
	Swaps: 0
	File system inputs: 0
	File system outputs: 33536
	Socket messages sent: 0
	Socket messages received: 0
	Signals delivered: 0
	Page size (bytes): 4096
	Exit status: 0
Calculating pi over 100000 iterations using SCHED_RR with 4096 simultaneous processes...
	Command being timed: "sudo ./pi-sched 100000 SCHED_RR 12"
	User time (seconds): 24.81
	System time (seconds): 1.81
	Percent of CPU this job got: 190%
	Elapsed (wall clock) time (h:mm:ss or m:ss): 0:13.99
	Average shared text size (kbytes): 0
	Average unshared data size (kbytes): 0
	Average stack size (kbytes): 0
	Average total size (kbytes): 0
	Maximum resident set size (kbytes): 4196
	Average resident set size (kbytes): 0
	Major (requiring I/O) page faults: 0
	Minor (reclaiming a frame) page faults: 150007
	Voluntary context switches: 8177
	Involuntary context switches: 24
	Swaps: 0
	File system inputs: 0
	File system outputs: 33536
	Socket messages sent: 0
	Socket messages received: 0
	Signals delivered: 0
	Page size (bytes): 4096
	Exit status: 0
Copying 102400 bytes in blocks of 1024 from rwinput to rwoutput
using SCHED_OTHER with 4096 simultaneous processes...
	Command being timed: "sudo ./rw 102400 1024 rwinput rwoutput 12
SCHED_OTHER"
	User time (seconds): 0.14
	System time (seconds): 34.61
	Percent of CPU this job got: 50%
	Elapsed (wall clock) time (h:mm:ss or m:ss): 1:09.13
	Average shared text size (kbytes): 0
	Average unshared data size (kbytes): 0
	Average stack size (kbytes): 0
	Average total size (kbytes): 0
	Maximum resident set size (kbytes): 4276
	Average resident set size (kbytes): 0
	Major (requiring I/O) page faults: 0
	Minor (reclaiming a frame) page faults: 152058
	Voluntary context switches: 1162827
	Involuntary context switches: 280383
	Swaps: 0
	File system inputs: 0
	File system outputs: 5091520
	Socket messages sent: 0
	Socket messages received: 0
	Signals delivered: 0
	Page size (bytes): 4096
	Exit status: 0
Copying 102400 bytes in blocks of 1024 from rwinput to rwoutput
using SCHED_FIFO with 4096 simultaneous processes...
	Command being timed: "sudo ./rw 102400 1024 rwinput rwoutput 12 SCHED_FIFO"
	User time (seconds): 0.30
	System time (seconds): 102.78
	Percent of CPU this job got: 84%
	Elapsed (wall clock) time (h:mm:ss or m:ss): 2:02.31
	Average shared text size (kbytes): 0
	Average unshared data size (kbytes): 0
	Average stack size (kbytes): 0
	Average total size (kbytes): 0
	Maximum resident set size (kbytes): 4216
	Average resident set size (kbytes): 0
	Major (requiring I/O) page faults: 0
	Minor (reclaiming a frame) page faults: 139803
	Voluntary context switches: 2726746
	Involuntary context switches: 39
	Swaps: 0
	File system inputs: 0
	File system outputs: 8569144
	Socket messages sent: 0
	Socket messages received: 0
	Signals delivered: 0
	Page size (bytes): 4096
	Exit status: 0
Copying 102400 bytes in blocks of 1024 from rwinput to rwoutput
using SCHED_RR with 4096 simultaneous processes...
	Command being timed: "sudo ./rw 102400 1024 rwinput rwoutput 12 SCHED_RR"
	User time (seconds): 0.22
	System time (seconds): 77.87
	Percent of CPU this job got: 66%
	Elapsed (wall clock) time (h:mm:ss or m:ss): 1:58.33
	Average shared text size (kbytes): 0
	Average unshared data size (kbytes): 0
	Average stack size (kbytes): 0
	Average total size (kbytes): 0
	Maximum resident set size (kbytes): 4252
	Average resident set size (kbytes): 0
	Major (requiring I/O) page faults: 0
	Minor (reclaiming a frame) page faults: 143935
	Voluntary context switches: 2775705
	Involuntary context switches: 19
	Swaps: 0
	File system inputs: 0
	File system outputs: 7427072
	Socket messages sent: 0
	Socket messages received: 0
	Signals delivered: 0
	Page size (bytes): 4096
	Exit status: 0
Running mixed CPU/IO bound process
using SCHED_OTHER with 4096 simultaneous processes...
	Command being timed: "sudo ./mixed 102400 1024 rwinput rwoutput 12
SCHED_OTHER 100000"
	User time (seconds): 21.32
	System time (seconds): 36.47
	Percent of CPU this job got: 55%
	Elapsed (wall clock) time (h:mm:ss or m:ss): 1:44.46
	Average shared text size (kbytes): 0
	Average unshared data size (kbytes): 0
	Average stack size (kbytes): 0
	Average total size (kbytes): 0
	Maximum resident set size (kbytes): 4200
	Average resident set size (kbytes): 0
	Major (requiring I/O) page faults: 0
	Minor (reclaiming a frame) page faults: 176623
	Voluntary context switches: 1248965
	Involuntary context switches: 323636
	Swaps: 0
	File system inputs: 0
	File system outputs: 5203440
	Socket messages sent: 0
	Socket messages received: 0
	Signals delivered: 0
	Page size (bytes): 4096
	Exit status: 0
Running mixed CPU/IO bound process
using SCHED_FIFO with 4096 simultaneous processes...
	Command being timed: "sudo ./mixed 102400 1024 rwinput rwoutput 12
SCHED_FIFO 100000"
	User time (seconds): 31.31
	System time (seconds): 84.80
	Percent of CPU this job got: 77%
	Elapsed (wall clock) time (h:mm:ss or m:ss): 2:29.78
	Average shared text size (kbytes): 0
	Average unshared data size (kbytes): 0
	Average stack size (kbytes): 0
	Average total size (kbytes): 0
	Maximum resident set size (kbytes): 4188
	Average resident set size (kbytes): 0
	Major (requiring I/O) page faults: 0
	Minor (reclaiming a frame) page faults: 184871
	Voluntary context switches: 3172366
	Involuntary context switches: 22
	Swaps: 0
	File system inputs: 0
	File system outputs: 9274960
	Socket messages sent: 0
	Socket messages received: 0
	Signals delivered: 0
	Page size (bytes): 4096
	Exit status: 0
Running mixed CPU/IO bound process
using SCHED_RR with 4096 simultaneous processes...
	Command being timed: "sudo ./mixed 102400 1024 rwinput rwoutput 12
SCHED_RR 100000"
	User time (seconds): 30.94
	System time (seconds): 70.28
	Percent of CPU this job got: 62%
	Elapsed (wall clock) time (h:mm:ss or m:ss): 2:42.72
	Average shared text size (kbytes): 0
	Average unshared data size (kbytes): 0
	Average stack size (kbytes): 0
	Average total size (kbytes): 0
	Maximum resident set size (kbytes): 4196
	Average resident set size (kbytes): 0
	Major (requiring I/O) page faults: 0
	Minor (reclaiming a frame) page faults: 180756
	Voluntary context switches: 2147220
	Involuntary context switches: 49
	Swaps: 0
	File system inputs: 0
	File system outputs: 8176656
	Socket messages sent: 0
	Socket messages received: 0
	Signals delivered: 0
	Page size (bytes): 4096
	Exit status: 0
user@cu-cs-vm:~/Dropbox/3753/PA4/pa3$
```

APPENDIX B: ALL CODE

-mixed.c-

```
/*
 * File: pi-sched.c
 * Author: Andy Sayler
 * Project: CSCI 3753 Programming Assignment 3
 * Create Date: 2012/03/07
 * Modify Date: 2012/03/09
 * Description:
 *         This file contains a simple program for statistically
 *      calculating pi using a specific scheduling policy.
 */

 /*
 * File: rw.c
 * Author: Andy Sayler
 * Project: CSCI 3753 Programming Assignment 3
 * Create Date: 2012/03/19
 * Modify Date: 2012/03/20
 * Description: A small i/o bound program to copy N bytes from an input
 *              file to an output file. May read the input file multiple
 *              times if N is larger than the size of the input file.
 */

 /* Include Flags */
#define _GNU_SOURCE

/* System Includes */
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <sched.h>
#include <math.h>
#include <fcntl.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>

#define DEFAULT_ITERATIONS 1000000
#define RADIUS (RAND_MAX / 2)
#define DEFAULT_PROCESSES 1
#define MAXFILENAMELENGTH 80
#define DEFAULT_INPUTFILENAME "rwinput"
#define DEFAULT_OUTPUTFILENAMEBASE "rwoutput"
#define DEFAULT_BLOCKSIZE 1024
#define DEFAULT_TRANSFERSIZE 1024*100
#define DEFAULT_PROCESSES 1

inline double dist(double x0, double y0, double x1, double y1){
   return sqrt(pow((x1-x0),2) + pow((y1-y0),2));
}

inline double zeroDist(double x, double y){
   return dist(0, 0, x, y);
}
```

```
int main(int argc, char* argv[]){

    int rv;
    int inputFD;
    int outputFD;
    int status;
    int policy;
    long processes;
    struct sched_param param;
    char inputFilename[MAXFILENAMELENGTH];
    char outputFilename[MAXFILENAMELENGTH];
    char outputFilenameBase[MAXFILENAMELENGTH];

    long i;
    long iterations;
    double x, y;
    double inCircle = 0.0;
    double inSquare = 0.0;
    double pCircle = 0.0;
    double piCalc = 0.0;

    ssize_t transfersize = 0;
    ssize_t blocksize = 0;
    char* transferBuffer = NULL;
    ssize_t buffersize;

    ssize_t bytesRead = 0;
    ssize_t totalBytesRead = 0;
    int totalReads = 0;
    ssize_t bytesWritten = 0;
    ssize_t totalBytesWritten = 0;
    int totalWrites = 0;
    int inputFileResets = 0;

    /* Process program arguments to select run-time parameters */
    /* Set supplied transfer size or default if not supplied */
    if(argc < 2){
        transfersize = DEFAULT_TRANSFERSIZE;
    }
    else{
        transfersize = atol(argv[1]);
        if(transfersize < 1){
            fprintf(stderr, "Bad transfersize value\n");
            exit(EXIT_FAILURE);
        }
    }
    /* Set supplied block size or default if not supplied */
    if(argc < 3){
        blocksize = DEFAULT_BLOCKSIZE;
    }
    else{
            blocksize = atol(argv[2]);
            if(blocksize < 1){
                fprintf(stderr, "Bad blocksize value\n");
                exit(EXIT_FAILURE);
            }
    }
    /* Set supplied input filename or default if not supplied */
    if(argc < 4){
            if(strnlen(DEFAULT_INPUTFILENAME, MAXFILENAMELENGTH) >=
MAXFILENAMELENGTH){
                fprintf(stderr, "Default input filename too long\n");
                exit(EXIT_FAILURE);
            }
            strncpy(inputFilename, DEFAULT_INPUTFILENAME,
MAXFILENAMELENGTH);
    }
    else{
            if(strnlen(argv[3], MAXFILENAMELENGTH) >= MAXFILENAMELENGTH){
                fprintf(stderr, "Input filename too long\n");
                exit(EXIT_FAILURE);
            }
            strncpy(inputFilename, argv[3], MAXFILENAMELENGTH);
    }
    /* Set supplied output filename base or default if not supplied */
    if(argc < 5){
            if(strnlen(DEFAULT_OUTPUTFILENAMEBASE, MAXFILENAMELENGTH) >=
MAXFILENAMELENGTH){
                fprintf(stderr, "Default output filename base too long\n");
                exit(EXIT_FAILURE);
            }
            strncpy(outputFilenameBase, DEFAULT_OUTPUTFILENAMEBASE,
MAXFILENAMELENGTH);
    }
    else{
            if(strnlen(argv[4], MAXFILENAMELENGTH) >= MAXFILENAMELENGTH){
                fprintf(stderr, "Output filename base is too long\n");
                exit(EXIT_FAILURE);
            }
            strncpy(outputFilenameBase, argv[4], MAXFILENAMELENGTH);
    }
    /* Set default policy if not supplied */
    if(argc < 7){
            policy = SCHED_OTHER;
    }
    /* Set policy if supplied */
    if(argc > 6){
            if(!strcmp(argv[6], "SCHED_OTHER")){
                policy = SCHED_OTHER;
            }
            else if(!strcmp(argv[6], "SCHED_FIFO")){
                policy = SCHED_FIFO;
            }
```

```
        else if(!strcmp(argv[6], "SCHED_RR")){
            policy = SCHED_RR;
        }
        else{
            fprintf(stderr, "Unhandeled scheduling policy\n");
            exit(EXIT_FAILURE);
        }
}
/* Set default iterations if not supplied */
if(argc < 8){
        iterations = DEFAULT_ITERATIONS;
}
/* Set iterations if supplied */
if(argc > 7){
        iterations = atol(argv[7]);
        if(iterations < 1){
            fprintf(stderr, "Bad iterations value\n");
            exit(EXIT_FAILURE);
        }
}
/* Set process to max prioty for given scheduler */
param.sched_priority = sched_get_priority_max(policy);
/* Set new scheduler policy */
fprintf(stdout, "Current Scheduling Policy: %d\n", sched_getscheduler(0));
fprintf(stdout, "Setting Scheduling Policy to: %d\n", policy);
if(sched_setscheduler(0, policy, &param)){
        perror("Error setting scheduler policy");
        exit(EXIT_FAILURE);
}
fprintf(stdout, "New Scheduling Policy: %d\n", sched_getscheduler(0));

/* Modification for forks */
if(argc < 6){
        processes = DEFAULT_PROCESSES;
        }

        /* Modification for forks */
if(argc > 5){
                processes = atol(argv[5]);
                if(processes < 1){
            fprintf(stderr, "Bad processes value\n");
            exit(EXIT_FAILURE);
                }
        }

        /* Modification for forks */
int f;
int g;
for(f=0; f<processes; f++){
                if(fork() == 0){
                        execve(argv[1],argv, NULL);
                }
        }
        for(g=0; g<processes; g++){
                waitpid(0, &status, 0);
        }

/* Confirm blocksize is multiple of and less than transfersize*/
if(blocksize > transfersize){
        fprintf(stderr, "blocksize can not exceed transfersize\n");
        exit(EXIT_FAILURE);
}
if(transfersize % blocksize){
        fprintf(stderr, "blocksize must be multiple of transfersize\n");
        exit(EXIT_FAILURE);
}

/* Allocate buffer space */
buffersize = blocksize;
if(!(transferBuffer = malloc(buffersize*sizeof(*transferBuffer)))){
        perror("Failed to allocate transfer buffer");
        exit(EXIT_FAILURE);
}

/* Open Input File Descriptor in Read Only mode */
if((inputFD = open(inputFilename, O_RDONLY | O_SYNC)) < 0){
        perror("Failed to open input file");
        exit(EXIT_FAILURE);
}

/* Open Output File Descriptor in Write Only mode with standard permissions*/
rv = snprintf(outputFilename, MAXFILENAMELENGTH, "%s-%d",
                      outputFilenameBase, getpid());
if(rv > MAXFILENAMELENGTH){
        fprintf(stderr, "Output filenmae length exceeds limit of %d characters.\n",
                MAXFILENAMELENGTH);
        exit(EXIT_FAILURE);
}
else if(rv < 0){
        perror("Failed to generate output filename");
        exit(EXIT_FAILURE);
}
if((outputFD =
        open(outputFilename,
             O_WRONLY | O_CREAT | O_TRUNC | O_SYNC,
             S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH)) < 0){
        perror("Failed to open output file");
        exit(EXIT_FAILURE);
}

/* Print Status */
fprintf(stdout, "Reading from %s and writing to %s\n",
            inputFilename, outputFilename);
```

```
/* Read from input file and write to output file*/
do{
        /* Read transfersize bytes from input file*/
        bytesRead = read(inputFD, transferBuffer, buffersize);
        if(bytesRead < 0){
            perror("Error reading input file");
            exit(EXIT_FAILURE);
        }
        else{
            totalBytesRead += bytesRead;
            totalReads++;
        }

        /* If all bytes were read, write to output file*/
        if(bytesRead == blocksize){
            bytesWritten = write(outputFD, transferBuffer, bytesRead);
            if(bytesWritten < 0){
                    perror("Error writing output file");
                    exit(EXIT_FAILURE);
            }
            else{
                    totalBytesWritten += bytesWritten;
                    totalWrites++;
            }
        }
        /* Otherwise assume we have reached the end of the input file and reset */
        else{
            if(lseek(inputFD, 0, SEEK_SET)){
                    perror("Error resetting to beginning of file");
                    exit(EXIT_FAILURE);
            }
            inputFileResets++;
        }

}while(totalBytesWritten < transfersize);

/* Output some possibly helpfull info to make it seem like we were doing stuff */
fprintf(stdout, "Read:    %zd bytes in %d reads\n",
            totalBytesRead, totalReads);
fprintf(stdout, "Written: %zd bytes in %d writes\n",
            totalBytesWritten, totalWrites);
fprintf(stdout, "Read input file in %d pass%s\n",
            (inputFileResets + 1), (inputFileResets ? "es" : ""));
fprintf(stdout, "Processed %zd bytes in blocks of %zd bytes\n",
            transfersize, blocksize);

/* Free Buffer */
free(transferBuffer);

/* Close Output File Descriptor */
if(close(outputFD)){
        perror("Failed to close output file");
```

```
            exit(EXIT_FAILURE);
    }

    /* Close Input File Descriptor */
    if(close(inputFD)){
            perror("Failed to close input file");
            exit(EXIT_FAILURE);
    }

    /* Calculate pi using statistical methode across all iterations*/
    for(i=0; i<iterations; i++){
            x = (random() % (RADIUS * 2)) - RADIUS;
            y = (random() % (RADIUS * 2)) - RADIUS;
            if(zeroDist(x,y) < RADIUS){
                inCircle++;
            }
            inSquare++;
    }

    /* Finish calculation */
    pCircle = inCircle/inSquare;
    piCalc = pCircle * 4.0;

    /* Print result */
    fprintf(stdout, "pi = %f\n", piCalc);

    return EXIT_SUCCESS;
}
```

-pi.c-

```
/*
 * File: pi.c
 * Author: Andy Sayler
 * Project: CSCI 3753 Programming Assignment 3
 * Create Date: 2012/03/07
 * Modify Date: 2012/03/09
 * Description:
 *         This file contains a simple program for statistically
 *      calculating pi.
 */

/* Local Includes */
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <errno.h>

/* Local Defines */
#define DEFAULT_ITERATIONS 1000000
#define RADIUS (RAND_MAX / 2)

/* Local Functions */
```

```
inline double dist(double x0, double y0, double x1, double y1){
   return sqrt(pow((x1-x0),2) + pow((y1-y0),2));
}

inline double zeroDist(double x, double y){
   return dist(0, 0, x, y);
}

int main(int argc, char* argv[]){

   long i;
   long iterations;
   double x, y;
   double inCircle = 0.0;
   double inSquare = 0.0;
   double pCircle = 0.0;
   double piCalc = 0.0;

   /* Process program arguments to select iterations */
   /* Set default iterations if not supplied */
   if(argc < 2){
         iterations = DEFAULT_ITERATIONS;
   }
   /* Set iterations if supplied */
   else{
         iterations = atol(argv[1]);
         if(iterations < 1){
            fprintf(stderr, "Bad iterations value\n");
            exit(EXIT_FAILURE);
         }
   }

   /* Calculate pi using statistical methode across all iterations*/
   for(i=0; i<iterations; i++){
         x = (random() % (RADIUS * 2)) - RADIUS;
         y = (random() % (RADIUS * 2)) - RADIUS;
         if(zeroDist(x,y) < RADIUS){
            inCircle++;
         }
         inSquare++;
   }

   /* Finish calculation */
   pCircle = inCircle/inSquare;
   piCalc = pCircle * 4.0;

   /* Print result */
   fprintf(stdout, "pi = %f\n", piCalc);

   return 0;
}
```

-rw.c-
```
/*
 * File: rw.c
 * Author: Andy Sayler
 * Project: CSCI 3753 Programming Assignment 3
 * Create Date: 2012/03/19
 * Modify Date: 2012/03/20
 * Description: A small i/o bound program to copy N bytes from an input
 *              file to an output file. May read the input file multiple
 *              times if N is larger than the size of the input file.
 */

/* Include Flags */
#define _GNU_SOURCE

/* System Includes */
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <sched.h>
#include <fcntl.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>

/* Local Defines */
#define MAXFILENAMELENGTH 80
#define DEFAULT_INPUTFILENAME "rwinput"
#define DEFAULT_OUTPUTFILENAMEBASE "rwoutput"
#define DEFAULT_BLOCKSIZE 1024
#define DEFAULT_TRANSFERSIZE 1024*100
#define DEFAULT_PROCESSES 1

int main(int argc, char* argv[]){

    int rv;
    int inputFD;
    int outputFD;
    int status;
    int policy;
    long processes;
    struct sched_param param;
    char inputFilename[MAXFILENAMELENGTH];
    char outputFilename[MAXFILENAMELENGTH];
    char outputFilenameBase[MAXFILENAMELENGTH];

    ssize_t transfersize = 0;
    ssize_t blocksize = 0;
    char* transferBuffer = NULL;
    ssize_t buffersize;
    ssize_t bytesRead = 0;
    ssize_t totalBytesRead = 0;
    int totalReads = 0;
    ssize_t bytesWritten = 0;
    ssize_t totalBytesWritten = 0;
    int totalWrites = 0;
    int inputFileResets = 0;

    /* Process program arguments to select run-time parameters */
    /* Set supplied transfer size or default if not supplied */
    if(argc < 2){
        transfersize = DEFAULT_TRANSFERSIZE;
    }
    else{
        transfersize = atol(argv[1]);
        if(transfersize < 1){
            fprintf(stderr, "Bad transfersize value\n");
            exit(EXIT_FAILURE);
        }
    }
    /* Set supplied block size or default if not supplied */
    if(argc < 3){
        blocksize = DEFAULT_BLOCKSIZE;
    }
    else{
        blocksize = atol(argv[2]);
        if(blocksize < 1){
            fprintf(stderr, "Bad blocksize value\n");
            exit(EXIT_FAILURE);
        }
    }
    /* Set supplied input filename or default if not supplied */
    if(argc < 4){
        if(strnlen(DEFAULT_INPUTFILENAME, MAXFILENAMELENGTH) >=
MAXFILENAMELENGTH){
            fprintf(stderr, "Default input filename too long\n");
            exit(EXIT_FAILURE);
        }
        strncpy(inputFilename, DEFAULT_INPUTFILENAME,
MAXFILENAMELENGTH);
    }
    else{
        if(strnlen(argv[3], MAXFILENAMELENGTH) >= MAXFILENAMELENGTH){
            fprintf(stderr, "Input filename too long\n");
            exit(EXIT_FAILURE);
        }
        strncpy(inputFilename, argv[3], MAXFILENAMELENGTH);
    }
    /* Set supplied output filename base or default if not supplied */
    if(argc < 5){
        if(strnlen(DEFAULT_OUTPUTFILENAMEBASE, MAXFILENAMELENGTH) >=
MAXFILENAMELENGTH){
            fprintf(stderr, "Default output filename base too long\n");
            exit(EXIT_FAILURE);
        }
        strncpy(outputFilenameBase, DEFAULT_OUTPUTFILENAMEBASE,
MAXFILENAMELENGTH);
    }
    else{
        if(strnlen(argv[4], MAXFILENAMELENGTH) >= MAXFILENAMELENGTH){
            fprintf(stderr, "Output filename base is too long\n");
            exit(EXIT_FAILURE);
        }
        strncpy(outputFilenameBase, argv[4], MAXFILENAMELENGTH);
    }
    /* Set default policy if not supplied */
    if(argc < 7){
        policy = SCHED_OTHER;
    }
    /* Set policy if supplied */
    if(argc > 6){
        if(!strcmp(argv[6], "SCHED_OTHER")){
            policy = SCHED_OTHER;
        }
        else if(!strcmp(argv[6], "SCHED_FIFO")){
            policy = SCHED_FIFO;
        }
        else if(!strcmp(argv[6], "SCHED_RR")){
            policy = SCHED_RR;
        }
        else{
            fprintf(stderr, "Unhandeled scheduling policy\n");
            exit(EXIT_FAILURE);
        }
    }
    /* Set process to max prioty for given scheduler */
    param.sched_priority = sched_get_priority_max(policy);
    /* Set new scheduler policy */
    fprintf(stdout, "Current Scheduling Policy: %d\n", sched_getscheduler(0));
    fprintf(stdout, "Setting Scheduling Policy to: %d\n", policy);
    if(sched_setscheduler(0, policy, &param)){
        perror("Error setting scheduler policy");
        exit(EXIT_FAILURE);
    }
    fprintf(stdout, "New Scheduling Policy: %d\n", sched_getscheduler(0));

    /* Modification for forks */
    if(argc < 6){
        processes = DEFAULT_PROCESSES;
        }

        /* Modification for forks */
```

```
if(argc > 5){
                processes = atol(argv[5]);
                if(processes < 1){
            fprintf(stderr, "Bad processes value\n");
            exit(EXIT_FAILURE);
                }
        }

        /* Modification for forks */
int f;
int g;
for(f=0; f<processes; f++){
                if(fork() == 0){
                        execve(argv[1],argv, NULL);
                }
        }
        for(g=0; g<processes; g++){
                waitpid(0, &status, 0);
        }

/* Confirm blocksize is multiple of and less than transfersize*/
if(blocksize > transfersize){
        fprintf(stderr, "blocksize can not exceed transfersize\n");
        exit(EXIT_FAILURE);
}
if(transfersize % blocksize){
        fprintf(stderr, "blocksize must be multiple of transfersize\n");
        exit(EXIT_FAILURE);
}

/* Allocate buffer space */
buffersize = blocksize;
if(!(transferBuffer = malloc(buffersize*sizeof(*transferBuffer)))){
        perror("Failed to allocate transfer buffer");
        exit(EXIT_FAILURE);
}

/* Open Input File Descriptor in Read Only mode */
if((inputFD = open(inputFilename, O_RDONLY | O_SYNC)) < 0){
        perror("Failed to open input file");
        exit(EXIT_FAILURE);
}

/* Open Output File Descriptor in Write Only mode with standard permissions*/
rv = snprintf(outputFilename, MAXFILENAMELENGTH, "%s-%d",
                 outputFilenameBase, getpid());
if(rv > MAXFILENAMELENGTH){
        fprintf(stderr, "Output filenmae length exceeds limit of %d characters.\n",
                MAXFILENAMELENGTH);
        exit(EXIT_FAILURE);
}
else if(rv < 0){
        perror("Failed to generate output filename");
        exit(EXIT_FAILURE);
}
if((outputFD =
        open(outputFilename,
             O_WRONLY | O_CREAT | O_TRUNC | O_SYNC,
             S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH)) < 0){
        perror("Failed to open output file");
        exit(EXIT_FAILURE);
}

/* Print Status */
fprintf(stdout, "Reading from %s and writing to %s\n",
            inputFilename, outputFilename);

/* Read from input file and write to output file*/
do{
        /* Read transfersize bytes from input file*/
        bytesRead = read(inputFD, transferBuffer, buffersize);
        if(bytesRead < 0){
            perror("Error reading input file");
            exit(EXIT_FAILURE);
        }
        else{
            totalBytesRead += bytesRead;
            totalReads++;
        }

        /* If all bytes were read, write to output file*/
        if(bytesRead == blocksize){
            bytesWritten = write(outputFD, transferBuffer, bytesRead);
            if(bytesWritten < 0){
                    perror("Error writing output file");
                    exit(EXIT_FAILURE);
            }
            else{
                    totalBytesWritten += bytesWritten;
                    totalWrites++;
            }
        }
        /* Otherwise assume we have reached the end of the input file and reset */
        else{
            if(lseek(inputFD, 0, SEEK_SET)){
                    perror("Error resetting to beginning of file");
                    exit(EXIT_FAILURE);
            }
            inputFileResets++;
        }

}while(totalBytesWritten < transfersize);

/* Output some possibly helpfull info to make it seem like we were doing stuff */
```

```
    fprintf(stdout, "Read:    %zd bytes in %d reads\n",
            totalBytesRead, totalReads);
    fprintf(stdout, "Written: %zd bytes in %d writes\n",
            totalBytesWritten, totalWrites);
    fprintf(stdout, "Read input file in %d pass%s\n",
            (inputFileResets + 1), (inputFileResets ? "es" : ""));
    fprintf(stdout, "Processed %zd bytes in blocks of %zd bytes\n",
            transfersize, blocksize);

    /* Free Buffer */
    free(transferBuffer);

    /* Close Output File Descriptor */
    if(close(outputFD)){
        perror("Failed to close output file");
        exit(EXIT_FAILURE);
    }

    /* Close Input File Descriptor */
    if(close(inputFD)){
        perror("Failed to close input file");
        exit(EXIT_FAILURE);
    }

    return EXIT_SUCCESS;
}
```

-MakeFile-

```
CC = gcc
CFLAGS = -c -g -Wall -Wextra
LFLAGS = -g -Wall -Wextra


INPUTFILESIZEMEGABYTES = 1

KILO = 1024
MEGA = $(shell echo $(KILO)\*$(KILO) | bc)
INPUTFILESIZEBYTES = $(shell echo $(MEGA)\*$(INPUTFILESIZEMEGABYTES) |
bc)
INPUTBLOCKSIZEBYTES = $(KILO)
INPUTBLOCKS = $(shell echo $(INPUTFILESIZEBYTES)\/
$(INPUTBLOCKSIZEBYTES) | bc)

.PHONY: all clean

all: pi pi-sched rw mixed

pi: pi.o
	$(CC) $(LFLAGS) $^ -o $@ -lm

pi-sched: pi-sched.o
```

```
	$(CC) $(LFLAGS) $^ -o $@ -lm

rw: rw.o rwinput
	$(CC) $(LFLAGS) rw.o -o $@ -lm

mixed: mixed.o rwinput
	$(CC) $(LFLAGS) mixed.o -o $@ -lm

pi.o: pi.c
	$(CC) $(CFLAGS) $<

pi-sched.o: pi-sched.c
	$(CC) $(CFLAGS) $<

rw.o: rw.c
	$(CC) $(CFLAGS) $<

mixed.o: mixed.c
	$(CC) $(CFLAGS) $<

rwinput: Makefile
	dd if=/dev/urandom of=./rwinput bs=$(INPUTBLOCKSIZEBYTES) count=
$(INPUTBLOCKS)

clean: testclean
	rm -f pi pi-sched rw mixed
	rm -f rwinput
	rm -f *.o
	rm -f *~
	rm -f handout/*~
	rm -f handout/*.log
	rm -f handout/*.aux

testclean:
	rm -f rwoutput*
	rm -f output*
```

-pi_sched.c-

```
/*
 * File: pi-sched.c
 * Author: Andy Sayler
 * Project: CSCI 3753 Programming Assignment 3
 * Create Date: 2012/03/07
 * Modify Date: 2012/03/09
 * Description:
 * 	This file contains a simple program for statistically
 *      calculating pi using a specific scheduling policy.
 */

/* Local Includes */
#include <stdlib.h>
```

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <errno.h>
#include <sched.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <unistd.h>

#define DEFAULT_ITERATIONS 1000000
#define RADIUS (RAND_MAX / 2)
#define DEFAULT_PROCESSES 1

inline double dist(double x0, double y0, double x1, double y1){
    return sqrt(pow((x1-x0),2) + pow((y1-y0),2));
}

inline double zeroDist(double x, double y){
    return dist(0, 0, x, y);
}

int main(int argc, char* argv[]){


    long i;
    long iterations;
    long processes;
    int status;
    struct sched_param param;
    int policy;
    double x, y;
    double inCircle = 0.0;
    double inSquare = 0.0;
    double pCircle = 0.0;
    double piCalc = 0.0;

    /* Process program arguments to select iterations and policy */
    /* Set default iterations if not supplied */
    if(argc < 2){
            iterations = DEFAULT_ITERATIONS;
    }
    /* Set default policy if not supplied */
    if(argc < 3){
            policy = SCHED_OTHER;
    }
    /* Modification for forks */
    if(argc < 4){
            processes = DEFAULT_PROCESSES;
            }
    /* Set iterations if supplied */
    if(argc > 1){
            iterations = atol(argv[1]);
```

```
        if(iterations < 1){
            fprintf(stderr, "Bad iterations value\n");
            exit(EXIT_FAILURE);
        }
}
/* Set policy if supplied */
if(argc > 2){
        if(!strcmp(argv[2], "SCHED_OTHER")){
            policy = SCHED_OTHER;
        }
        else if(!strcmp(argv[2], "SCHED_FIFO")){
            policy = SCHED_FIFO;
        }
        else if(!strcmp(argv[2], "SCHED_RR")){
            policy = SCHED_RR;
        }
        else{
            fprintf(stderr, "Unhandeled scheduling policy\n");
            exit(EXIT_FAILURE);
        }
}
/* Modification for forks */
if(argc > 3){
                processes = atol(argv[3]);
                if(processes < 1){
            fprintf(stderr, "Bad processes value\n");
            exit(EXIT_FAILURE);
                }
        }

/* Set process to max prioty for given scheduler */
param.sched_priority = sched_get_priority_max(policy);

/* Set new scheduler policy */
fprintf(stdout, "Current Scheduling Policy: %d\n", sched_getscheduler(0));
fprintf(stdout, "Setting Scheduling Policy to: %d\n", policy);
if(sched_setscheduler(0, policy, &param)){
        perror("Error setting scheduler policy");
        exit(EXIT_FAILURE);
}
fprintf(stdout, "New Scheduling Policy: %d\n", sched_getscheduler(0));

/* Modification for forks */
int f;
int g;
for(f=0; f<processes; f++){
                if(fork() == 0){
                        execve(argv[1],argv, NULL);
                }
        }
        for(g=0; g<processes; g++){
                waitpid(0, &status, 0);
        }

    /* Calculate pi using statistical methode across all iterations*/
    for(i=0; i<iterations; i++){
        x = (random() % (RADIUS * 2)) - RADIUS;
        y = (random() % (RADIUS * 2)) - RADIUS;
        if(zeroDist(x,y) < RADIUS){
            inCircle++;
        }
        inSquare++;
    }

    /* Finish calculation */
    pCircle = inCircle/inSquare;
    piCalc = pCircle * 4.0;

    /* Print result */
    fprintf(stdout, "pi = %f\n", piCalc);

    return 0;
}
```

-TestScript-

```
#/!/bin/bash

#File: testscript
#Author: Andy Sayler
#Project: CSCI 3753 Programming Assignment 3
#Create Date: 2012/03/09
#Modify Date: 2012/03/21
#Description:
#       A simple bash script to run a signle copy of each test case
#       and gather the relevent data.

ITERATIONS=100000 #100000000
SML=6
MED=9
LRG=12
BYTESTOCOPY=102400
BLOCKSIZE=1024
INPUTNAME=rwinput
OUTPUTNAME=rwoutput
INPUTFILES=names1.txt names2.txt names3.txt
RESULTS=results.txt
MAKE="make -s"

echo Building code...
$MAKE clean
$MAKE

echo Starting test runs...
echo Calculating pi over $ITERATIONS iterations using SCHED_OTHER with 64
simultaneous processes...
/usr/bin/time -v sudo ./pi-sched $ITERATIONS SCHED_OTHER $SML >> output

echo Calculating pi over $ITERATIONS iterations using SCHED_FIFO with 64
simultaneous processes...
/usr/bin/time -v sudo ./pi-sched $ITERATIONS SCHED_FIFO $SML >> output

echo Calculating pi over $ITERATIONS iterations using SCHED_RR with 64 simultaneous
processes...
/usr/bin/time -v sudo ./pi-sched $ITERATIONS SCHED_RR $SML >> output

echo Copying $BYTESTOCOPY bytes in blocks of $BLOCKSIZE from rwinput to rwoutput
echo using SCHED_OTHER with 64 simultaneous processes...
/usr/bin/time -v sudo ./rw $BYTESTOCOPY $BLOCKSIZE $INPUTNAME
$OUTPUTNAME $SML SCHED_OTHER >> output

echo Copying $BYTESTOCOPY bytes in blocks of $BLOCKSIZE from rwinput to rwoutput
echo using SCHED_FIFO with 64 simultaneous processes...
/usr/bin/time -v sudo ./rw $BYTESTOCOPY $BLOCKSIZE $INPUTNAME
$OUTPUTNAME $SML SCHED_FIFO >> output

echo Copying $BYTESTOCOPY bytes in blocks of $BLOCKSIZE from rwinput to rwoutput
echo using SCHED_RR with 64 simultaneous processes...
/usr/bin/time -v sudo ./rw $BYTESTOCOPY $BLOCKSIZE $INPUTNAME
$OUTPUTNAME $SML SCHED_RR >> output

echo Running mixed CPU/IO bound process
echo using SCHED_OTHER with 64 simultaneous processes...
/usr/bin/time -v sudo ./mixed $BYTESTOCOPY $BLOCKSIZE $INPUTNAME
$OUTPUTNAME $SML SCHED_OTHER $ITERATIONS >> output

echo Running mixed CPU/IO bound process
echo using SCHED_FIFO with 64 simultaneous processes...
/usr/bin/time -v sudo ./mixed $BYTESTOCOPY $BLOCKSIZE $INPUTNAME
$OUTPUTNAME $SML SCHED_FIFO $ITERATIONS >> output

echo Running mixed CPU/IO bound process
echo using SCHED_RR with 64 simultaneous processes...
/usr/bin/time -v sudo ./mixed $BYTESTOCOPY $BLOCKSIZE $INPUTNAME
$OUTPUTNAME $SML SCHED_RR $ITERATIONS >> output

echo Calculating pi over $ITERATIONS iterations using SCHED_OTHER with 512
simultaneous processes...
/usr/bin/time -v sudo ./pi-sched $ITERATIONS SCHED_OTHER $MED >> output

echo Calculating pi over $ITERATIONS iterations using SCHED_FIFO with 512
simultaneous processes...
/usr/bin/time -v sudo ./pi-sched $ITERATIONS SCHED_FIFO $MED >> output

echo Calculating pi over $ITERATIONS iterations using SCHED_RR with 512 simultaneous
processes...
```

```
/usr/bin/time -v sudo ./pi-sched $ITERATIONS SCHED_RR $MED >> output

echo Copying $BYTESTOCOPY bytes in blocks of $BLOCKSIZE from rwinput to rwoutput
echo using SCHED_OTHER with 512 simultaneous processes...
/usr/bin/time -v sudo ./rw $BYTESTOCOPY $BLOCKSIZE $INPUTNAME
$OUTPUTNAME $MED SCHED_OTHER >> output

echo Copying $BYTESTOCOPY bytes in blocks of $BLOCKSIZE from rwinput to rwoutput
echo using SCHED_FIFO with 512 simultaneous processes...
/usr/bin/time -v sudo ./rw $BYTESTOCOPY $BLOCKSIZE $INPUTNAME
$OUTPUTNAME $MED SCHED_FIFO >> output

echo Copying $BYTESTOCOPY bytes in blocks of $BLOCKSIZE from rwinput to rwoutput
echo using SCHED_RR with 512 simultaneous processes...
/usr/bin/time -v sudo ./rw $BYTESTOCOPY $BLOCKSIZE $INPUTNAME
$OUTPUTNAME $MED SCHED_RR >> output

echo Running mixed CPU/IO bound process
echo using SCHED_OTHER with 512 simultaneous processes...
/usr/bin/time -v sudo ./mixed $BYTESTOCOPY $BLOCKSIZE $INPUTNAME
$OUTPUTNAME $MED SCHED_OTHER $ITERATIONS >> output

echo Running mixed CPU/IO bound process
echo using SCHED_FIFO with 512 simultaneous processes...
/usr/bin/time -v sudo ./mixed $BYTESTOCOPY $BLOCKSIZE $INPUTNAME
$OUTPUTNAME $MED SCHED_FIFO $ITERATIONS >> output

echo Running mixed CPU/IO bound process
echo using SCHED_RR with 512 simultaneous processes...
/usr/bin/time -v sudo ./mixed $BYTESTOCOPY $BLOCKSIZE $INPUTNAME
$OUTPUTNAME $MED SCHED_RR $ITERATIONS >> output

echo Calculating pi over $ITERATIONS iterations using SCHED_OTHER with 4096
simultaneous processes...
/usr/bin/time -v sudo ./pi-sched $ITERATIONS SCHED_OTHER $LRG >> output

echo Calculating pi over $ITERATIONS iterations using SCHED_FIFO with 4096
simultaneous processes...
/usr/bin/time -v sudo ./pi-sched $ITERATIONS SCHED_FIFO $LRG >> output

echo Calculating pi over $ITERATIONS iterations using SCHED_RR with 4096
simultaneous processes...
/usr/bin/time -v sudo ./pi-sched $ITERATIONS SCHED_RR $LRG >> output

echo Copying $BYTESTOCOPY bytes in blocks of $BLOCKSIZE from rwinput to rwoutput
echo using SCHED_OTHER with 4096 simultaneous processes...
/usr/bin/time -v sudo ./rw $BYTESTOCOPY $BLOCKSIZE $INPUTNAME
$OUTPUTNAME $LRG SCHED_OTHER >> output

echo Copying $BYTESTOCOPY bytes in blocks of $BLOCKSIZE from rwinput to rwoutput
echo using SCHED_FIFO with 4096 simultaneous processes...
```

```
/usr/bin/time -v sudo ./rw $BYTESTOCOPY $BLOCKSIZE $INPUTNAME
$OUTPUTNAME $LRG SCHED_FIFO >> output

echo Copying $BYTESTOCOPY bytes in blocks of $BLOCKSIZE from rwinput to rwoutput
echo using SCHED_RR with 4096 simultaneous processes...
/usr/bin/time -v sudo ./rw $BYTESTOCOPY $BLOCKSIZE $INPUTNAME
$OUTPUTNAME $LRG SCHED_RR >> output

echo Running mixed CPU/IO bound process
echo using SCHED_OTHER with 4096 simultaneous processes...
/usr/bin/time -v sudo ./mixed $BYTESTOCOPY $BLOCKSIZE $INPUTNAME
$OUTPUTNAME $LRG SCHED_OTHER $ITERATIONS >> output

echo Running mixed CPU/IO bound process
echo using SCHED_FIFO with 4096 simultaneous processes...
/usr/bin/time -v sudo ./mixed $BYTESTOCOPY $BLOCKSIZE $INPUTNAME
$OUTPUTNAME $LRG SCHED_FIFO $ITERATIONS >> output

echo Running mixed CPU/IO bound process
echo using SCHED_RR with 4096 simultaneous processes...
/usr/bin/time -v sudo ./mixed $BYTESTOCOPY $BLOCKSIZE $INPUTNAME
$OUTPUTNAME $LRG SCHED_RR $ITERATIONS >> output
```

-util.c-

```
/*
 * File: util.c
 * Author: Andy Sayler
 * Project: CSCI 3753 Programming Assignment 2
 * Create Date: 2012/02/01
 * Modify Date: 2012/02/01
 * Description:
 *        This file contains declarations of utility functions for
 *     Programming Assignment 2.
 *
 */

#include "util.h"

int dnslookup(const char* hostname, char* firstIPstr, int maxSize){

    /* Local vars */
    struct addrinfo* headresult = NULL;
    struct addrinfo* result = NULL;
    struct sockaddr_in* ipv4sock = NULL;
    struct sockaddr_in6* ipv6sock = NULL;
    struct in_addr* ipv4addr = NULL;
    struct in6_addr* ipv6addr = NULL;
    char ipv4str[INET_ADDRSTRLEN];
    char ipv6str[INET6_ADDRSTRLEN];
    char ipstr[INET6_ADDRSTRLEN];
    int addrError = 0;
```

```
   /* DEBUG: Print Hostname*/
#ifdef UTIL_DEBUG
   fprintf(stderr, "%s\n", hostname);
#endif

   /* Lookup Hostname */
   addrError = getaddrinfo(hostname, NULL, NULL, &headresult);
   if(addrError){
         fprintf(stderr, "Error looking up Address: %s\n",
                  gai_strerror(addrError));
         return UTIL_FAILURE;
   }
   /* Loop Through result Linked List */
   for(result=headresult; result != NULL; result = result->ai_next){
         /* Extract IP Address and Convert to String */
         if(result->ai_addr->sa_family == AF_INET){
            /* IPv4 Address Handling */
            ipv4sock = (struct sockaddr_in*)(result->ai_addr);
            ipv4addr = &(ipv4sock->sin_addr);
            if(!inet_ntop(result->ai_family, ipv4addr,
                           ipv4str, sizeof(ipv4str))){
                  perror("Error Converting IP to String");
                  return UTIL_FAILURE;
            }
#ifdef UTIL_DEBUG
            fprintf(stdout, "%s\n", ipv4str);
#endif
            strncpy(ipstr, ipv4str, sizeof(ipstr));
            ipstr[sizeof(ipstr)-1] = '\0';
         }
         else if(result->ai_addr->sa_family == AF_INET6){
            /* IPv6 Handling */
            ipv6sock = (struct sockaddr_in6*)(result->ai_addr);
            ipv6addr = &(ipv6sock->sin6_addr);
            if(!inet_ntop(result->ai_family, ipv6addr,
                           ipv6str, sizeof(ipv6str))){
                  perror("Error Converting IP to String");
                  return UTIL_FAILURE;
         }
#ifdef UTIL_DEBUG
                        fprintf(stdout, "IPv6 Address: Not Handled\n");
#endif
                        strncpy(ipstr, ipv6str, sizeof(ipstr));
                        ipstr[sizeof(ipstr)-1] = '\0';
                  }
                  else{
            /* Unhandlded Protocol Handling */
#ifdef UTIL_DEBUG
                        fprintf(stdout, "Unknown Protocol: Not Handled\n");
#endif
                        strncpy(ipstr, "UNHANDELED", sizeof(ipstr));
                        ipstr[sizeof(ipstr)-1] = '\0';
```

```
		}
	/* Save First IP Address */
		if(result==headresult){
			strncpy(firstIPstr, ipstr, maxSize);
			firstIPstr[maxSize-1] = '\0';
		}
   }

   /* Cleanup */
   freeaddrinfo(headresult);

   return UTIL_SUCCESS;
}
```

-util.h-

```
/*
 * File: util.h
 * Author: Andy Sayler
 * Project: CSCI 3753 Programming Assignment 2
 * Create Date: 2012/02/01
 * Modify Date: 2012/02/01
 * Description:
 * 	This file contains declarations of utility functions for
 *      Programming Assignment 2.
 *
 */

#ifndef UTIL_H
#define UTIL_H

/* Define the following to enable debug statments */
// #define UTIL_DEBUG

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>

#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>

#define UTIL_FAILURE -1
#define UTIL_SUCCESS 0
#define NUM_ADDR 50

/* Fuction to return the first IP address found
 * for hostname. IP address returned as string
 * firstIPstr of size maxsize
 */
int dnslookup(const char* hostname,
char* firstIPstr,
int maxSize);
```