Programming Assignment 3:
Investigating the Linux Scheduler

Edward Zhu
CSCI 3753 Operating Systems
University of Colorado at Boulder
March 27, Spring 2014

ABSTRACT

By creating 3 different benchmarks to test the performance of **CPU Bound** programs, **I/O bound** programs, and a **Mix of CPU and I/O Bound** programs it will provide succinct data in observing the behavior of the Linux Scheduler. The three different schedulers investigated were **First in-First out (FIFO)**, **Round Robin (RR)**, and **Completely Fair Scheduler (CFS)**. Testing happened on the CU Virtual Machine of Spring '14, which is a 64 bit machine running Ubuntu 12.04. The virtual machine was also running on top of a 64 bit version of Windows 7 with Intel VT-x hardware assisted virtualization enabled on its chip. Although most of the collected data did stay true to convention; they may have been slightly skewed data because of the virtual machine environment. The results indicate that CFS provides an advantage in turnaround time and CPU utilization, but has its disadvantages as context switch overhead is a bit higher than RR and FIFO.

INTRODUCTION

Each Linux scheduler provides different ways of utilizing CPU and IO processing. There were three test programs that tested FIFO, RR, and CFS with IO bound, CPU bound, and a Mix. Then each of the bound programs were repeated with different load amount to simulate how each test program would perform with light, medium, or heavy loads of processing. FIFO scheduling maintains a queue, and then picks the process that has spent to longest time in the queue (more simply put the first process to be pushed into the queue) and does not preempt any other processes. RR scheduling provides each process a certain runtime, and then preempts each, next process giving each process a turn to execute. CFS is rather similar to RR, but it gives certain time slices based on the priority and CPU process execution. CFS is currently the default scheduler for Linux and this project will test why that is and whether it is the correct scheduler for the job.

METHOD

As stated before, there will be three different configurations to test, each with their own subtests. FIFO, RR, and CFS scheduling will be tested at differently process loads for CPU bound, I/O bound and Mixed bound programs. The loads are simulated by forking each program to run 5, 70, and 300 simultaneous processes. For the CPU bound test program, the value of pi is calculated over a certain number of iterations to throttle the CPU. The IO bound test program reads from an input file and writes to an output file a certain amount of times, which utilizes the read and write system calls that will speak with the hard drive. The mixed test program has the same idea as the CPU bound test program, but the calculated value of pi and piCircle is written to an output file, which will utilize the CPU along with I/O devices. The Unix Time shell command was used to time each program. It gives the total time, system time, user time, CPU utilization, and also the number of voluntary or involuntary context switches. A shell script was created to run all the cases 5 times and pushed to a file for analyzing with Microsoft Excel.

RESULTS

Following are column graphs that compare each scheduler:

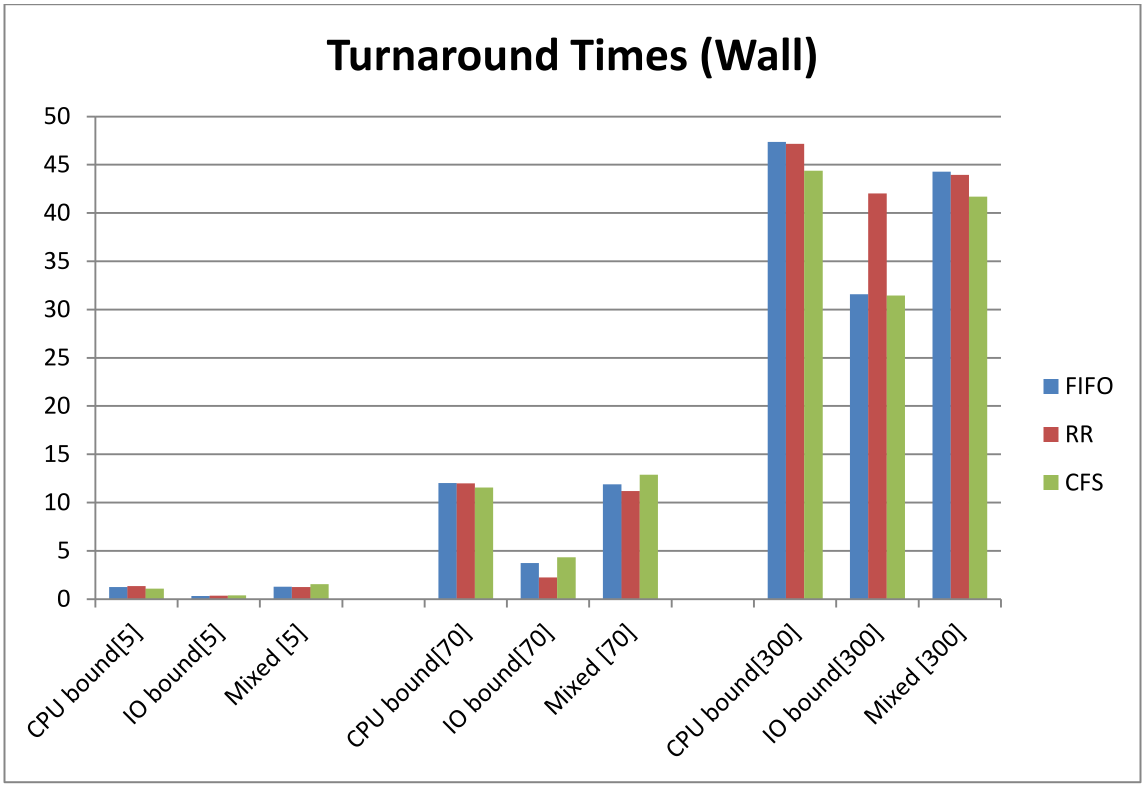

Figure 1

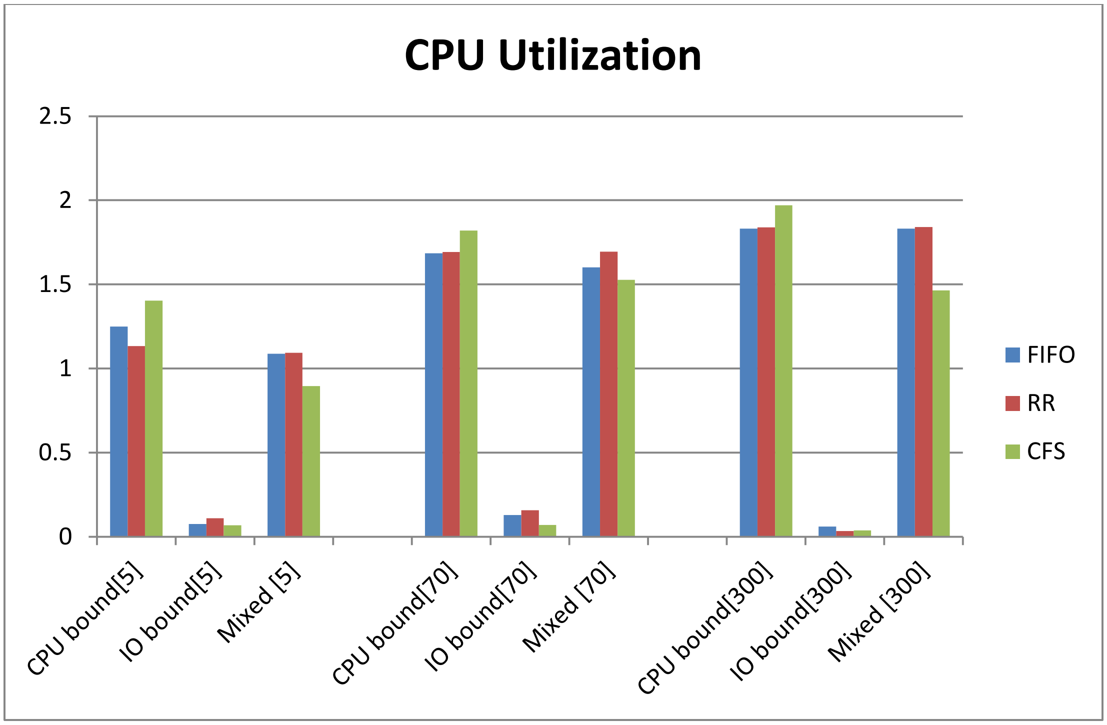

Figure 2

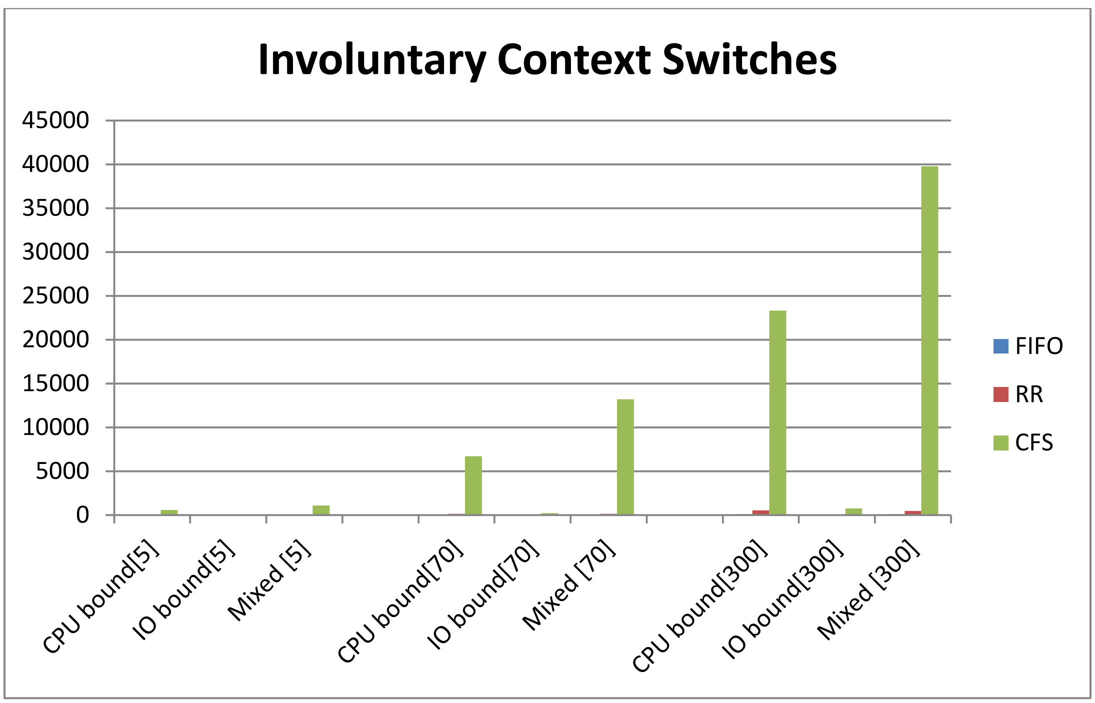

Figure 3

Following is the averaged data from all the testing of all three schedulers:

	FIFO					
	wall	user	system	CPU	i-switched	v-switched
CPU bound[5]	1.262	1.47	0.016	1.25	2	9.4
IO bound[5]	0.306	0.004	0.01	0.076	3.4	1163
Mixed [5]	1.284	1.284	0.028	1.088	5	10.6
CPU bound[70]	12.022	20.262	0.056	1.686	21.2	75
IO bound[70]	3.724	0.06	0.392	0.13	4.8	20693.4
Mixed [70]	11.902	18.96	0.174	1.602	20.2	94.8
CPU bound[300]	47.382	86.948	0.146	1.832	90	305.2
IO bound[300]	31.58	0.274	1.36	0.06	2.4	91798.4
Mixed [300]	44.274	81.124	0.23	1.832	84.2	395
	RR					
	wall	user	system	CPU	i-switched	v-switched
CPU bound[5]	1.342	1.474	0.008	1.134	15	10
IO bound[5]	0.356	0.002	0.032	0.11	1.6	1396.2
Mixed [5]	1.26	1.346	0.016	1.094	12.4	9.6
CPU bound[70]	11.99	20.288	0.038	1.692	119	84.4
IO bound[70]	2.236	0.066	0.292	0.158	2.2	20790.6
Mixed [70]	11.186	18.826	0.134	1.694	116.6	86.8
CPU bound[300]	47.186	86.928	0.156	1.84	537.4	427.2
IO bound[300]	42.046	0.31	1.314	0.034	3.2	93891.2
Mixed [300]	43.956	81.078	0.176	1.842	463.6	368.8
	CFS					
	wall	user	system	CPU	i-switched	v-switched
CPU bound[5]	1.092	1.456	0.028	1.404	573.4	16
IO bound[5]	0.398	0.004	0.018	0.068	36.6	1815.8
Mixed [5]	1.53	1.32	0.05	0.896	1069.6	12
CPU bound[70]	11.548	20.702	0.154	1.82	6708.8	140.2
IO bound[70]	4.344	0.068	0.188	0.07	221.2	23050.6
Mixed [70]	12.878	18.834	0.866	1.528	13206.2	148
CPU bound[300]	44.382	87.432	0.224	1.97	23319.2	599.2
IO bound[300]	31.458	0.352	0.884	0.038	751.2	102582.8
Mixed [300]	41.686	81.9	1.906	1.464	39781.4	627.2

Figure 4

ALALYSIS

After reviewing the data, it seems that CFS provides the best Wall time, or turnaround time, which will provide a better experience for everyday computing by the average human being. In most tests CFS appears to have a smaller wall time than all the other schedulers. Having a good turnaround time allows for a quick response time from the program when multi-tasking. Although CFS provides overall better response time, Figure 3 shows just how context switch heavy CFS is and how it appears to be linearly proportional to the number of simultaneous processes running. Involuntary context switches happened when you force processes to give up access to the CPU or to halt and switch to different process.

User and system data represent how many CPU-seconds a process is spent in User mode or in Kernal mode, respectively. From the data, it seems that User mode is more frequently used for CPU bound programs while the process goes into Kernal mode when it is an IO bound program.

If we were speaking about CPU utilization, where we want better usage of the CPU and don't care as much about response time, then RR or FIFO schedulers would be better suited for the task because they provide a better CPU utilization for CPU bound programs. Round Robin seems to scale slightly better than First in-First out, which mean it would most likely be a better scheduler by a slight amount in most situations.

Although the data seems to be conventionally correct, there was room for a few major concerns within these tests and experiment. Because the testing environment was in a virtual machine on top another operating system, the results could be slightly inaccurate and inconsistent. Running the test programs on a native Linux machine would have provided better results for the testing. Another concern would be that it doesn't accurately represent a day to day usage of computing because there are never true interrupts or cancelations on the processes, which happens in everyday computing. If this was taken into consideration, then that data might indicate a stronger depiction for CFS than FIFO or RR than the data calculated in this testing environment. Lastly, the limited amount of tests and variety might also add to some inaccuracy to the data. The tests only accounted for 5, 70, and 300 processes during a certain constrained, short period of time, which might not provide to most accurate data because of how processes are usually handled for everyday computing. These few issues could be the reason why there very well could be inaccuracies and skewing on the data.

CONCLUSION

These test programs conveyed that the CFS is the most reliable, overall scheduler for an average computing environment and confirms its default state in the majority of the latest Linux kernels. Although CFS is the all-around best, FIFO and RR do have their merits of a much lower context switch overheads and slightly better CPU utilization especially for CPU bound programs, but do fall inferior when it comes to turnaround time compared to CFS. In general, all three Linux schedulers have their strengths and weaknesses, but because response time is a vital component to the average user, CFS is the most suitable scheduler for most day to day computing.

REFERENCES

http://en.wikipedia.org/wiki/Scheduling_(computing)

http://oreilly.com/catalog/linuxkernel/chapter/ch10.html

http://en.wikipedia.org/wiki/Completely_Fair_Scheduler

https://www.linux.com/linux-man-pages

APPENDIX A – RAW DATA

	wall	user	system	CPU	i-switch	v-switch
FIFO HEAVY CPU	46.51	87	0.12	187%	92	305
	47.06	86.92	0.13	184%	91	306
	47.55	86.82	0.18	182%	90	305
	47.84	86.95	0.17	182%	89	305
	47.95	87.05	0.13	181%	88	305
	47.382	**86.948**	**0.146**	**1.832**	**90**	**305.2**
FIFO HEAVY IO	34.99	0.32	1.1	5%	4	95892
	26.66	0.22	1.64	7%	3	92527
	32.79	0.38	0.85	5%	2	88238
	31.54	0.26	1.59	5%	3	90889
	31.92	0.19	1.62	8%	0	91446
	31.58	**0.274**	**1.36**	**0.06**	**2.4**	**91798.4**
FIFO HEAVY MIXED	44.71	81.4	0.14	182%	85	306
	44.15	80.9	0.18	183%	84	305
	44.2	80.99	0.33	183%	84	305
	44.21	81.12	0.26	184%	83	305
	44.1	81.21	0.24	184%	85	754
	44.274	**81.124**	**0.23**	**1.832**	**84.2**	**395**
FIFO LIGHT CPU	1.51	1.46	0.03	98%	2	9
	0.9	1.49	0	165%	1	10
	1.5	1.47	0.01	99%	3	9
	1.51	1.45	0	98%	1	9
	0.89	1.48	0.04	165%	3	10
	1.262	**1.47**	**0.016**	**1.25**	**2**	**9.4**
FIFO LIGHT IO	0.29	0	0.02	9%	5	1123
	0.24	0.01	0	8%	4	1009
	0.25	0	0.01	8%	1	1009
	0.37	0.01	0	6%	3	1213
	0.38	0	0.02	7%	4	1461
	0.306	**0.004**	**0.01**	**0.076**	**3.4**	**1163**
FIFO LIGHT MIXED	1.07	1.07	0.02	129%	2	10
	1.43	1.43	0.05	98%	10	10
	1.37	1.37	0	98%	4	9
	1.17	1.17	0.07	121%	6	10
	1.38	1.38	0	98%	3	14
	1.284	**1.284**	**0.028**	**1.088**	**5**	**10.6**

FIFO MEDIUM CPU	12.05	20.26	0.02	168%	27	75
	12.08	20.28	0.05	168%	19	75
	11.97	20.24	0.06	169%	20	75
	12.01	20.26	0.08	169%	18	75
	12	20.27	0.07	169%	22	75
	12.022	**20.262**	**0.056**	**1.686**	**21.2**	**75**
FIFO MEDIUM IO	4.24	0.07	0.49	12%	10	21215
	2.27	0.09	0.19	13%	1	19308
	3.29	0.05	0.51	17%	7	20896
	5.76	0.06	0.28	6%	4	21162
	3.06	0.03	0.49	17%	2	20886
	3.724	**0.06**	**0.392**	**0.13**	**4.8**	**20693.4**
FIFO MEDIUM MIX	11.6	18.94	0.03	163%	20	75
	11.79	18.92	0.01	160%	17	75
	12.17	19.02	0.37	159%	23	75
	11.95	18.96	0.15	160%	19	75
	12	18.96	0.31	159%	22	174
	11.902	**18.96**	**0.174**	**1.602**	**20.2**	**94.8**
OTHER HEAVY CPU	44.4	87.34	0.21	197%	23319	600
	44.41	87.48	0.25	197%	23444	598
	44.38	87.47	0.24	197%	23120	600
	44.32	87.34	0.22	197%	23341	598
	44.4	87.53	0.2	197%	23372	600
	44.382	**87.432**	**0.224**	**1.97**	**23319.2**	**599.2**
OTHER HEAVY IO	31.02	0.35	0.89	4%	881	103379
	28.87	0.32	0.91	4%	700	103366
	31.03	0.35	0.9	4%	902	100904
	27.81	0.35	0.87	4%	655	103359
	38.56	0.39	0.85	3%	618	101906
	31.458	**0.352**	**0.884**	**0.038**	**751.2**	**102583**
OTHER HEAVY MIX	35.35	81.16	3.93	130%	59071	601
	45.86	82.12	1.83	87%	45862	614
	37.8	82.13	1.48	144%	34472	594
	45.08	82.12	1.22	184%	30193	622
	44.34	81.97	1.07	187%	29309	705
	41.686	**81.9**	**1.906**	**1.464**	**39781.4**	**627.2**

OTHER LIGHT CPU	1.36	1.44	0	106%	281	29
	1.13	1.43	0.04	129%	778	13
	0.79	1.46	0.01	185%	377	13
	0.96	1.49	0.02	158%	322	12
	1.22	1.46	0.07	124%	1109	13
	1.092	**1.456**	**0.028**	**1.404**	**573.4**	**16**
OTHER LIGHT IO	0.37	0.01	0.01	6%	14	1915
	0.31	0.01	0.02	11%	9	1854
	0.39	0	0.02	6%	106	1642
	0.45	0	0.02	6%	21	1777
	0.47	0	0.02	5%	33	1891
	0.398	**0.004**	**0.018**	**0.068**	**36.6**	**1815.8**
OTHER LIGHT MIXE	1.58	1.27	0.08	85%	1474	13
	1.56	1.31	0.04	86%	1234	14
	1.31	1.34	0.03	104%	671	11
	1.66	1.36	0.04	84%	748	11
	1.54	1.32	0.06	89%	1221	11
	1.53	**1.32**	**0.05**	**0.896**	**1069.6**	**12**
OTHER MEDIUM CI	13.63	20.83	0.26	154%	8056	139
	12.36	20.85	0.3	171%	7808	141
	10.74	20.92	0.06	195%	5818	140
	10.53	20.45	0.1	195%	6155	141
	10.48	20.46	0.05	195%	5707	140
	11.548	**20.702**	**0.154**	**1.82**	**6708.8**	**140.2**
OTHER MEDIUM IC	2.19	0.1	0.16	12%	246	22693
	2.28	0.04	0.16	9%	259	22670
	5.58	0.06	0.21	5%	198	23314
	4.84	0.08	0.19	5%	234	23321
	6.83	0.06	0.22	4%	169	23255
	4.344	**0.068**	**0.188**	**0.07**	**221.2**	**23050.6**
OTHER MEDIUM M	13.11	18.67	0.96	149%	17283	142
	13.04	18.91	0.84	151%	12757	141
	13.56	18.84	1.09	147%	13733	156
	12.89	18.87	0.8	152%	12400	155
	11.79	18.88	0.64	165%	9858	146
	12.878	**18.834**	**0.866**	**1.528**	**13206.2**	**148**

RR HEAVY CPU	47.78	86.82	0.22	182%	527	461
	47.9	86.95	0.13	181%	554	501
	47.99	87.02	0.16	181%	538	385
	46.11	86.95	0.14	188%	536	436
	46.15	86.9	0.13	188%	532	353
	47.186	**86.928**	**0.156**	**1.84**	**537.4**	**427.2**
RR HEAVY IO	29.41	0.38	0.89	4%	5	88374
	43.25	0.23	1.46	3%	8	95873
	51.96	0.36	1.06	2%	1	98570
	49.34	0.26	1.94	4%	1	93700
	36.27	0.32	1.22	4%	1	92939
	42.046	**0.31**	**1.314**	**0.034**	**3.2**	**93891.2**
RR HEAVY MIXED	44	81.25	0.12	184%	466	377
	43.9	80.94	0.27	184%	463	377
	43.82	80.86	0.13	184%	466	405
	44.23	81.32	0.26	184%	464	313
	43.83	81.02	0.1	185%	459	372
	43.956	**81.078**	**0.176**	**1.842**	**463.6**	**368.8**
RR LIGHT CPU	1.19	1.46	0	123%	12	10
	1.52	1.49	0.02	99%	20	10
	1.48	1.48	0	148%	13	12
	1.52	1.46	0.02	98%	12	9
	1	1.48	0	99%	18	9
	1.342	**1.474**	**0.008**	**1.134**	**15**	**10**
RR LIGHT IO	0.35	0	0.04	13%	2	1462
	0.23	0	0.01	8%	0	1011
	0.33	0.01	0.04	15%	3	1594
	0.41	0	0.03	9%	1	1323
	0.46	0	0.04	10%	2	1591
	0.356	**0.002**	**0.032**	**0.11**	**1.6**	**1396.2**
RR LIGHT MIXED	1.36	1.35	0	99%	12	9
	1.04	1.34	0	129%	12	11
	1.38	1.34	0.02	99%	15	9
	1.37	1.36	0	99%	12	9
	1.15	1.34	0.06	121%	11	10
	1.26	**1.346**	**0.016**	**1.094**	**12.4**	**9.6**

RR MEDIUM CPU	12.35	20.25	0.08	164%	123	77
	11.72	20.29	0	173%	117	104
	11.9	20.25	0.03	170%	118	77
	11.95	20.27	0.06	169%	117	86
	12.03	20.38	0.02	170%	120	78
	11.99	**20.288**	**0.038**	**1.692**	**119**	**84.4**
RR MEDIUM IO	2.83	0.06	0.34	14%	2	21198
	1.74	0.05	0.23	16%	0	21486
	1.96	0.06	0.21	14%	2	21101
	2.31	0.08	0.48	24%	5	21094
	2.34	0.08	0.2	11%	2	19074
	2.236	**0.066**	**0.292**	**0.158**	**2.2**	**20790.6**
RR MEDIUM MIZED	11.49	18.8	0.05	164%	116	84
	10.04	18.8	0.04	187%	109	77
	11.02	18.91	0.03	171%	113	86
	11.6	18.78	0.32	164%	125	104
	11.78	18.84	0.23	161%	120	83
	11.186	**18.826**	**0.134**	**1.694**	**116.6**	**86.8**

APPENDIX B – DESCIPTION OF SOURCE FILES

CPUbound.c – modified pi-sched.c file where the number of processes is forked right when the calculation of pi happens, making all the children calculate pi under the same parent, giving the same calculation, but the calculations is done in different loads.

IObound.c – modified rw.c file where the number of processes is forked when the output file is given a name and it ends when both the output and input files are closed. This allows each child process to create a new output file and read from input and write to its specific output file individually.

MIXEDbound.c – modified CPUbound.c file where when the values of pi and piCircle are calculated within the forked child process, it writes them to a junk.txt file. This allows utilization of both CPU and IO at the same time with a certain load of processes.

Makefile – simple make file that makes and cleans all object files, also clean the junk.txt file.

runscript – shell script that runes each type of test (total of 27 test) 5 times and writes them to a file for further analyzing.