Investigating the Linux Scheduler

Christopher Costello

March 24, 2014

CSCI 3753 - Operating Systems

University of Colorado at Boulder Spring 2014

Abstract

This work examines the FIFO, RR, BFS, and CFS scheduling algorithms in a common Linux environment. Each scheduler scheduled sets of 5, 60, and 160 CPU intensive, I/O intensive, and mixed programs. With the fastest average runtimes, highest CPU utilization, and exception efficiency, the Brain Fuck Scheduler is clearly the best scheduler for the machine tested. However, because of BFS' lack of scalability with 16 CPU machines and higher, CFS is the most well suited scheduler for the standard Linux kernel.

Introduction

Different Linux schedulers use different algorithms and some may perform better than others under various conditions. My test programs gathered data from First In First Out (FIFO) scheduler, Round Robin (RR) scheduler, Completely Fair Scheduler (CFS) and the Brain Fuck Scheduler (BFS) all working to schedule CPU, I/O intensive programs, as well as mixed programs. For consistency, each scheduler only scheduled instances of the same program. I gathered various benchmarks including wall time, CPU time, context switches, and CPU usage. From these figures I hope to draw conclusions about these four popular Linux schedulers.

Method

I used three programs as the focal point for my experimentation. The first program, *pi-sched.c*, is a CPU bound program that uses the statistical method to calculate pi in 10 million iterations. The code is capable of forking itself and changing its scheduler from command line arguments. On the other side of the spectrum is the I/O bound program, *rw.c*, which reads 25 megabytes in 256 kilobytes blocks from an input file and writes the same amount to an output file. This program also can set the scheduling policy and fork itself based on command line arguments. In the middle is *mixed.c*, which reads 1

kilobyte from an input file, calculates pi for 1 million iterations, and writes 1 kilobyte to an output file for 100 kilobytes.

I ran each program under the FIFO, Round Robin, CFS, and BFS scheduler each with 5 processes (Light), 60 processes (Medium) and 160 processes (Heavy). I used the Linux time command to gather metrics on the process execution. I then averaged the results of each combination over 3 runs. Metrics from each program were output to csv files which a java program then averaged and put to one common csv file. Data was then analyzed in Microsoft Excel.

The testing environment is on the CU-CS 64-bit Ubuntu virtual machine run on VM Ware Fusion from within a 13 inch Retina MacBook Pro from Late 2012 plugged in on AC power. The virtual machine gets 2 CPU cores, 30GB of disk and 2GB of RAM. All runs were launched from a single script with no other user programs running in the virtual machine or in the native Mac environment.

Results

From Figures 1 and 2 we see that BFS is generally faster across the board for Light and Medium process loads. FIFO is faster than all others for completing high volume I/O bound processes.

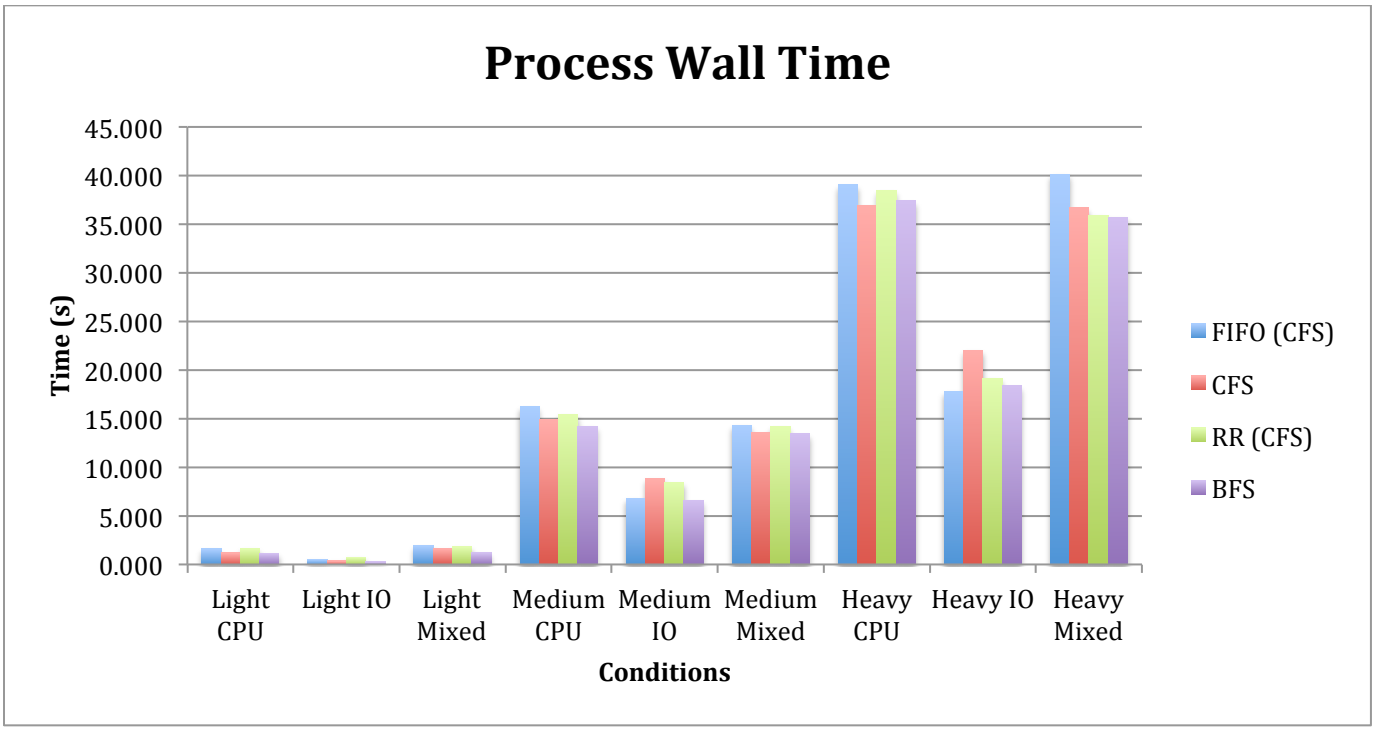

Figure 1

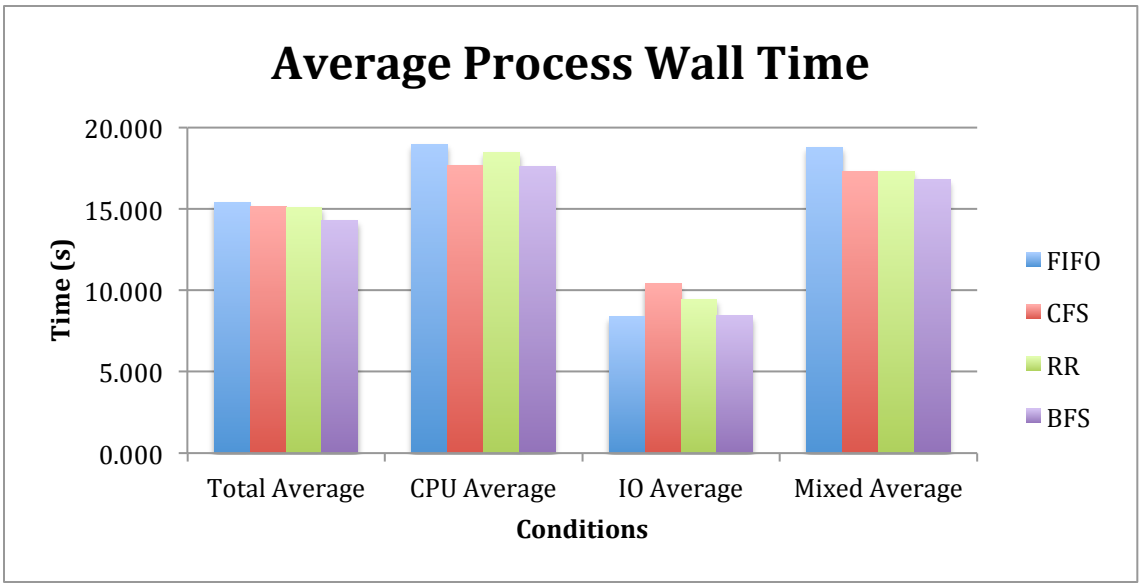

Figure 2

Figure 3 shows that processes scheduled by CFS and BFS generally use more of the CPU than FIFO or RR. Figures 5, 6, and 7 show that processes scheduled by CFS and BFS generally spend more total time executing on the CPU and incur more context switches.

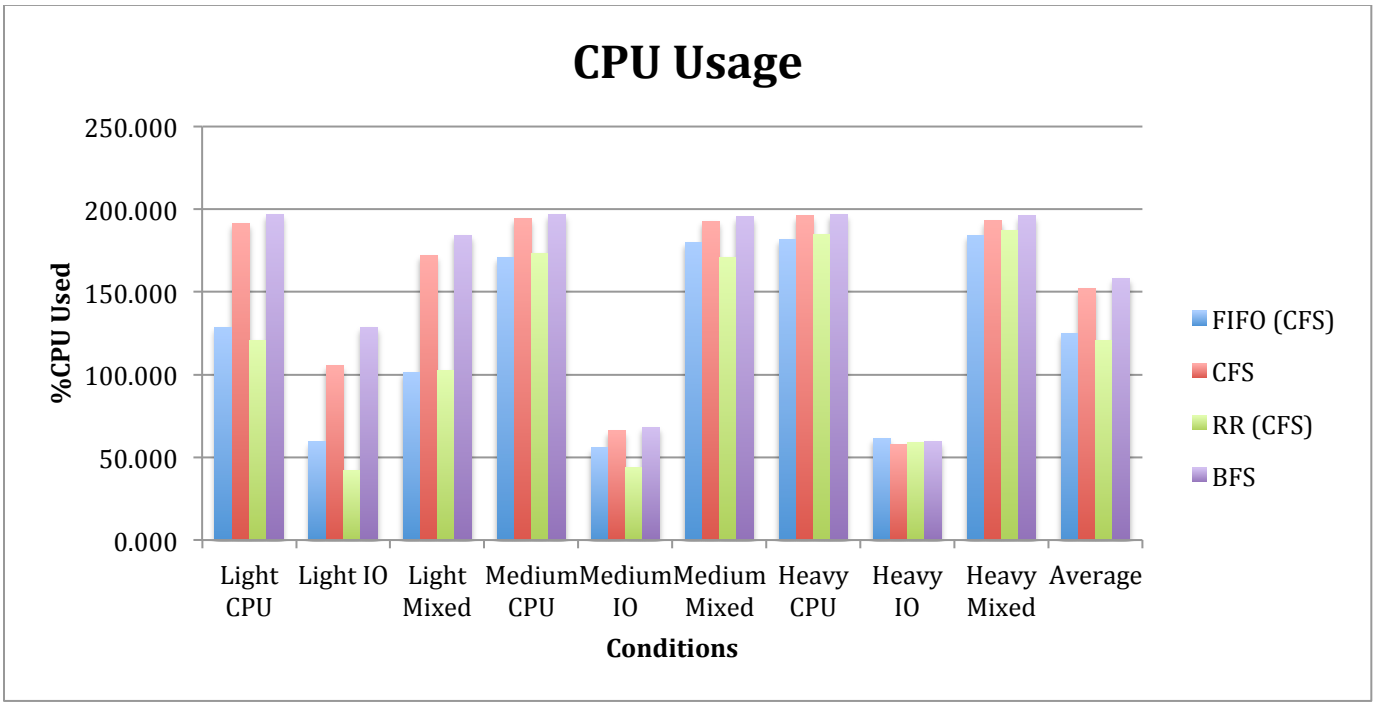

Figure 3

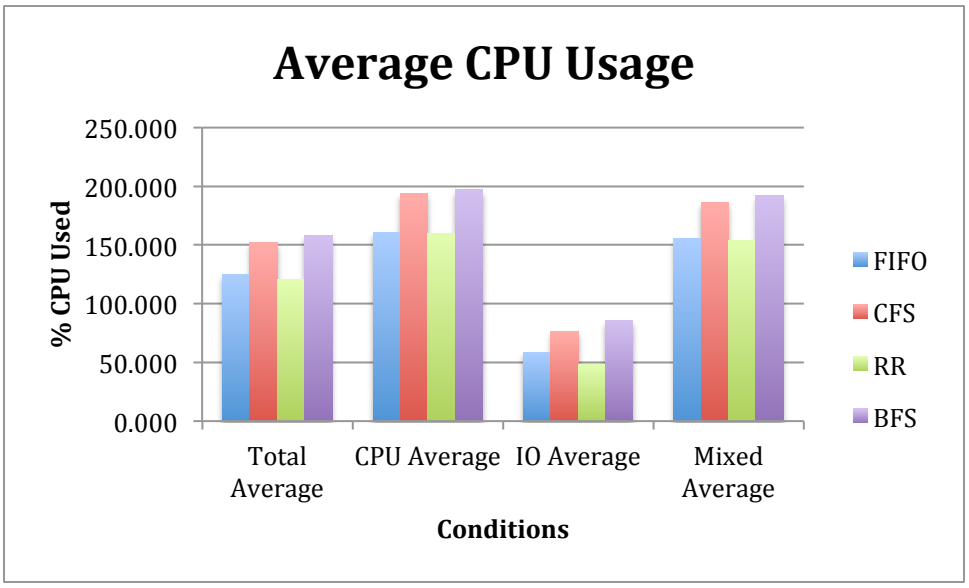

Figure 4

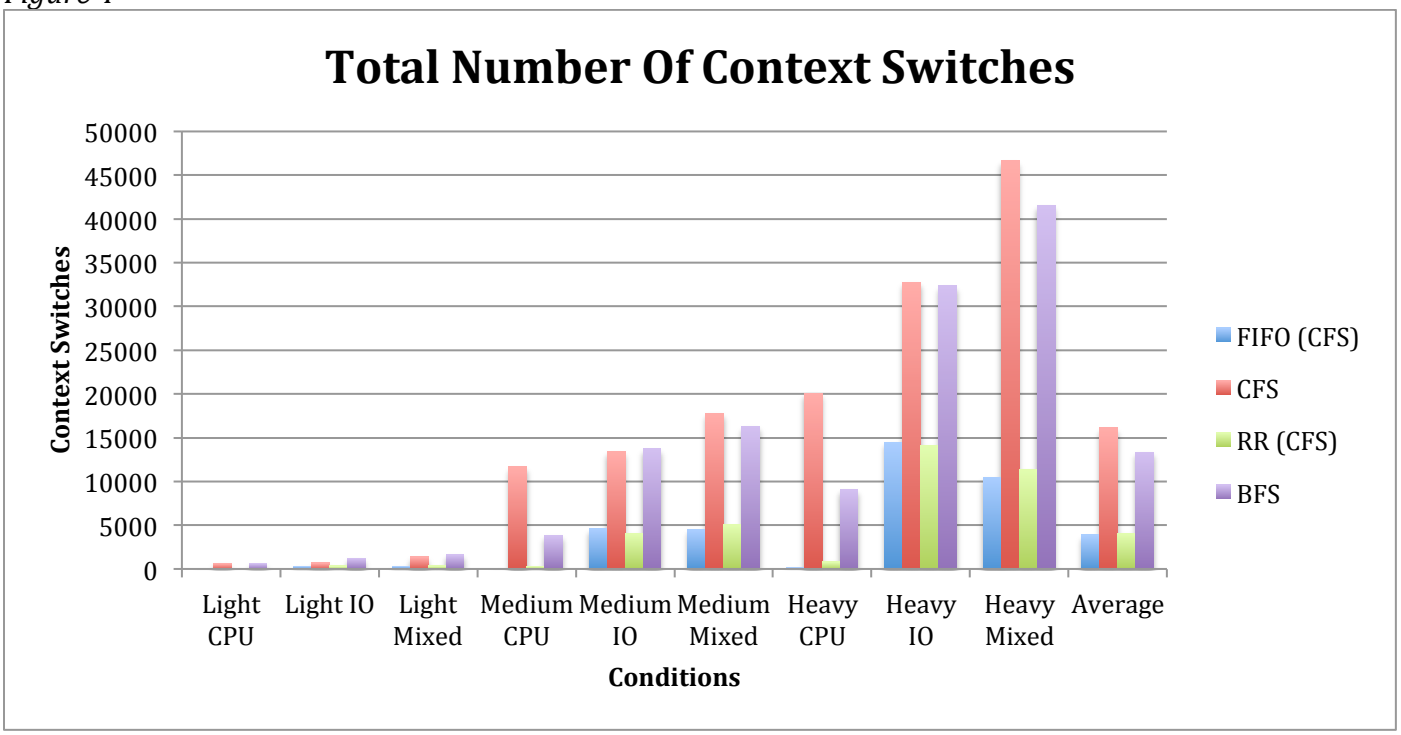

Figure 5

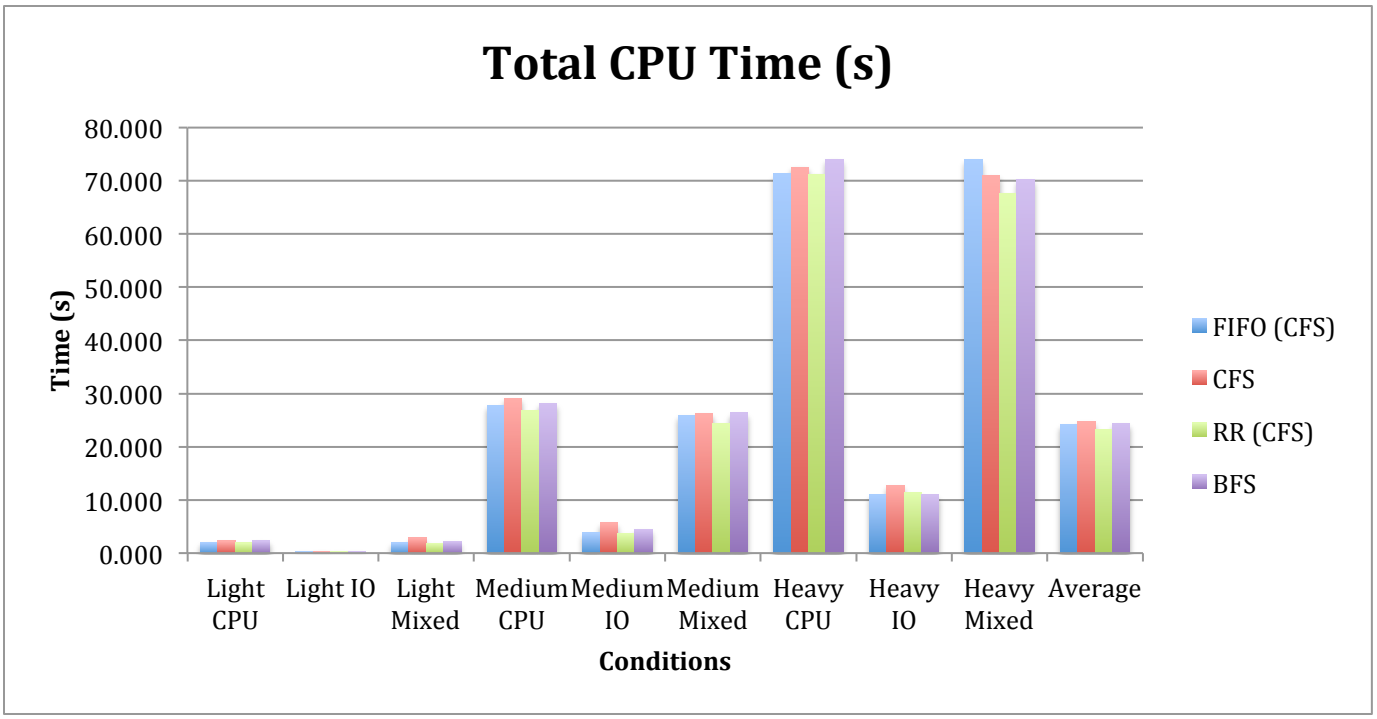

Figure 6

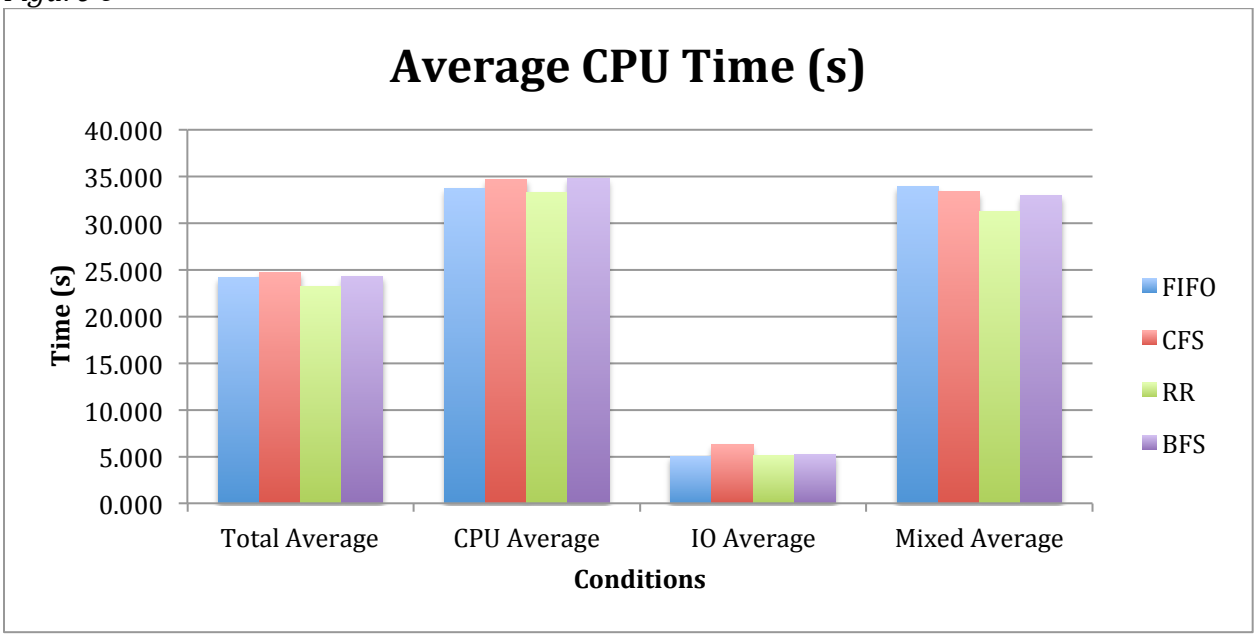

Figure 7

Analysis

The FIFO scheduler is the best suited for high volume I/O intensive processes. It has the fastest running time and the lowest CPU usage and therefore the highest efficiency. The Round Robin scheduler is best suited for mixed programs. On average it completes mixed processes in the same amount of time as CFS, but because of the

drastically lower CPU usage, it achieves a much higher efficiency. CFS excels in CPU bound processes because of superior run times over FIFO and RR across each level of process intensity despite having much higher context switch overhead. This means that CFS makes better use of the CPU than the other two schedulers because it achieved the same amount of work plus overhead in a shorter amount of time. BFS excels in all process types especially with light and medium loads. BFS achieves this with high CPU utilization and a modest amount of context switch overhead achieving high efficiency.

Each scheduler scales slightly differently. FIFO scales well for I/O bound processes because it keeps context switch overhead to a minimum. Processes will voluntarily give up the CPU enough for other processes to get time, so the scheduler does not need to incur extra overhead in preempting processes. It does not scale as well for CPU bound processes because turnaround time will suffer. CFS does not scale well in I/O bound processes because of context switch overhead. It preempts processes that already voluntarily give up the CPU so overhead becomes an issue as the numbers of processes grow. It does scale well with CPU bound processes, however. The variable time-slices maximize CPU efficiency. Round Robin scales decently. While it does not have the best running times for I/O or CPU bound processes, it does not lag behind the leaders. Round Robin does incur slightly more context switch overhead for I/O than FIFO, but not nearly to the extent of CFS or BFS. It also doesn't have the low run time for CPU bound processes that CFS does, but it does have much less overhead and doesn't lag far behind. BFS scales the best overall. It achieves the lowest runtimes and the highest CPU utilization compared to the others. It has similar trends to CFS, but has significantly less context switch overhead than CFS. However, as Kolivas notes, "a machine with 16 CPUS or more would start to have exponentially less performance"[1]. The computer used for this experiment only has 2 CPUs, which is what BFS was developed for. BFS would not perform well in an HPC environment.

Each scheduling policy has advantages and disadvantages. FIFO is good for keeping context switch overhead to a minimum, but suffers from response time when the queue of processes waiting for the CPU grows and processes executing don't voluntarily give up the CPU. CFS minimizes waiting time because of the variable time slices, but suffers from high context switch overhead. Unlike FIFO, Round Robin is "fairer" to smaller processes that are waiting behind large processes. Because of the same quantum for all processes, however, throughput may suffer. BFS has the highest CPU utilization but does suffer from high context switch overhead compared to FIFO and RR.

FIFO is well suited for an environment with multiple I/O bound processes. Context switch overhead is small because processes will voluntarily give up the CPU. Round Robin would excel in an environment where context switches are especially expensive yet response time is still important. CFS is a good general-purpose scheduler. It performs well in CPU bound and mixed environments, justifying its mainstream usage. BFS also performs well in general-purpose environments. It outperforms all other schedulers considered here and comes close to FIFO in heavy I/O process loads.

Conclusion

From this data, the BFS scheduler is by far the best scheduling algorithm. With its low running times, high CPU utilization, and modest context switch overhead, it outperforms all other schedulers. The only exception is the FIFO scheduler, which excels in high volume of I/O bound processes, yet BFS is not far behind. I would recommend that the Brain Fuck Scheduler come standard in typical (sub 16 CPU) Linux environments. However, the author of BFS would probably disagree. His answer to if he wants BFS in the mainline was simply "LOL"[1]. He further elaborates that BFS won't scale well beyond 16 CPU machines[1]. For this reason, CFS is the logical choice to encompass all computing environments the standard Linux kernel might be applied to.

References

[1] http://ck.kolivas.org/patches/bfs/bfs-faq.txt
[2] http://ck.wikia.com/wiki/BFS
[3] http://en.wikipedia.org/wiki/Completely_Fair_Scheduler
[4] http://inst.eecs.berkeley.edu/~cs162/sp11/sections/cs162-sp11-section5-answers.pdf

Appendix A

Row Labels	Wall Time	Total CPU Time	CPU Usage	Total Context Switch
FIFO				
Heavy				
CPU	39.06333333	71.35	182	171
IO	17.76333333	10.95666667	61.33333333	14454.66667
MIXED	40.08666667	74.03	184.3333333	10492.33333
Light	4.096666667	4.323333333	289.3333333	681.3333333
CPU	1.62	2.036666667	128.6666667	6.666666667
IO	0.533333333	0.316666667	59.33333333	337.3333333
MIXED	1.943333333	1.97	101.3333333	337.3333333
Medium	37.34333333	57.39666667	406.3333333	9140
CPU	16.23333333	27.76666667	170.6666667	60
IO	6.793333333	3.79	55.66666667	4585.666667
MIXED	14.31666667	25.84	180	4494.333333
CFS				
Heavy				
CPU	36.96333333	72.58666667	196	20058
IO	22	12.71333333	57.66666667	32778
MIXED	36.67333333	70.99333333	193	46626
Light	3.223333333	5.53	469.3333333	2728.333333
CPU	1.193333333	2.296666667	191.6666667	641
IO	0.38	0.386666667	105.3333333	711
MIXED	1.65	2.846666667	172.3333333	1376.333333
Medium	37.34333333	61.04666667	453.3333333	42819.66667
CPU	14.92	29.13	194.6666667	11661.66667
IO	8.816666667	5.663333333	66	13376.33333
MIXED	13.60666667	26.25333333	192.6666667	17781.66667
RR	135.73	208.9633333	1085	36442
Heavy	93.50666667	150.0866667	431.3333333	26345
CPU	38.41666667	71.24666667	185	818.6666667
IO	19.15	11.30666667	59	14123.66667
MIXED	35.94	67.53333333	187.3333333	11402.66667
Light	4.226666667	4.116666667	265.3333333	722.6666667
CPU	1.636666667	1.92	120.6666667	25.66666667
IO	0.75	0.31	42	340.3333333
MIXED	1.84	1.886666667	102.6666667	356.6666667
Medium	37.99666667	54.76	388.3333333	9374.333333
CPU	15.37666667	26.72	173.3333333	314
IO	8.426666667	3.71	44	4030.666667
MIXED	14.19333333	24.33	171	5029.666667

Row Labels	Wall Time	Total CPU Time	CPU Usage	Total Context Switch
FIFO	125.94	216.66	1414.666667	102977.3333
Heavy	89.28666667	153.5166667	457.3333333	71318.33333
CPU	37.15333333	73.49333333	197	5917.333333
IO	16.96666667	10.67666667	63.33333333	29029.66667
MIXED	35.16666667	69.34666667	197	36371.33333
Light	2.643333333	4.743333333	497.3333333	2774
CPU	1.12	2.126666667	189.3333333	386.3333333
IO	0.293333333	0.353333333	124	893
MIXED	1.23	2.263333333	184	1494.666667
Medium	34.01	58.4	460	28885
CPU	14.2	28.05666667	197	2554.333333
IO	6.61	4.406666667	67	12445
MIXED	13.2	25.93666667	196	13885.66667
CFS	128.47	218.9433333	1422.666667	120288.6667
Heavy	91.48666667	155.07	452.6666667	82936.66667
CPU	37.4	73.95	197	9065
IO	18.39	10.93333333	59.66666667	32337.66667
MIXED	35.69666667	70.18666667	196	41534
Light	2.676666667	4.93	509.6666667	3536
CPU	1.173333333	2.32	197	664
IO	0.276666667	0.35	128.6666667	1241
MIXED	1.226666667	2.26	184	1631
Medium	34.30666667	58.94333333	460.3333333	33816
CPU	14.23	28.1	197	3839
IO	6.623333333	4.496666667	68	13723.33333
MIXED	13.45333333	26.34666667	195.3333333	16253.66667
RR	126.36	215.3233333	1427.333333	101793.3333
Heavy	90.02	152.1466667	451	72185.33333
CPU	36.86	72.92333333	197	6040.666667
IO	18.33	10.52666667	57	29727
MIXED	34.83	68.69666667	197	36417.66667
Light	2.543333333	4.673333333	515	1967.666667
CPU	1.086666667	2.133333333	195.3333333	142
IO	0.246666667	0.336666667	138.3333333	728
MIXED	1.21	2.203333333	181.3333333	1097.666667
Medium	33.79666667	58.50333333	461.3333333	27640.33333
CPU	14.38333333	28.38	196.6666667	2364
IO	6.273333333	4.3	68.66666667	11255.33333
MIXED	13.14	25.82333333	196	14021

Appendix B

pi-sched.c

- Calculates pi using the statistical method using 10,000,000 iterations. Forks itself from number in command line arguments. Sets scheduler from command line arguments as well.

rw.c

- Reads 25 MB from a file, and writes 25 MB to a process specific output file in 256-kilobyte blocks. Forks itself from number in command line arguments. Sets scheduler from command line arguments as well.

mixed.c

- Reads 1 kilobyte from an input file, calculates pi for 1 million iterations, and writes 1 kilobyte to an output file for 100 kilobytes. Forks itself from number in command line arguments. Sets scheduler from command line arguments as well.