Luke Woodruff & Annie Kelly

CSCI 3753

Programming Assignment 3

11/4/2014

Linux Scheduler Experiment

Abstract

In this report we will be benchmarking three different scheduling algorithms: normal, round robin, and first in-first out. We will be testing these schedulers across three different program types: IO-Bound, CPU-Bound, and Mixed. In addition, we will run these programs and fork off a different number of processes each time: Low (5 processes), Medium (50 processes), High (200 processes). We will record our results using time to record the Wall, User, System, CPU, i-switched, and v-switched.

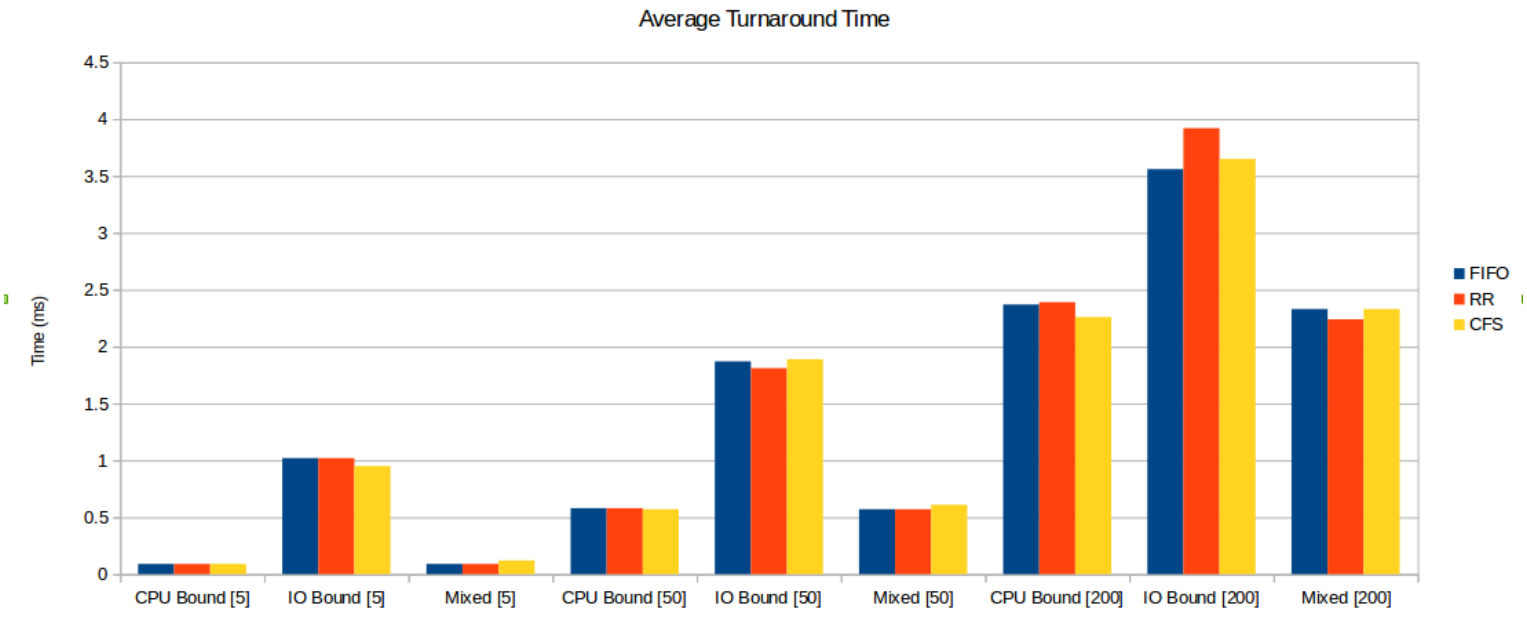
Introduction

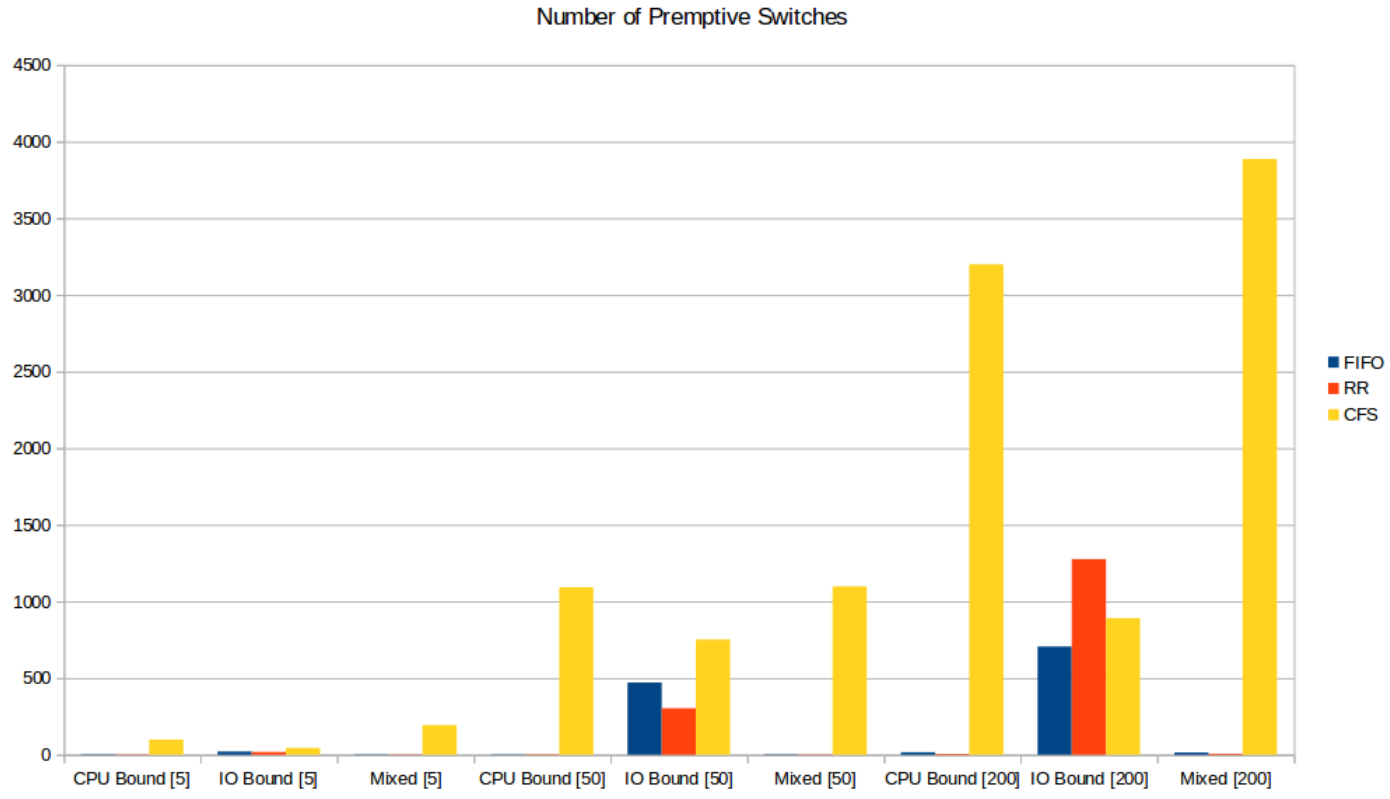
The purpose of this investigation is to probe into the behavior of the Linux scheduler. We looked at the performance of three schedulers. The first-in-first-out (FIFO) scheduler maintains a queue and picks processes from the front of the queue, that is those having been in the queue the longest, and does not preempt any process. Round Robin (RR) gives each process a quantum of runtime, then preemptively switches to the next process until all have gotten a turn. The Completely Fair Scheduler (CFS) is the default scheduler in Linux and, at a high level, is similar to the Round Robin scheduling, but it adjusts the time quantum based on the amount of CPU it previously used.

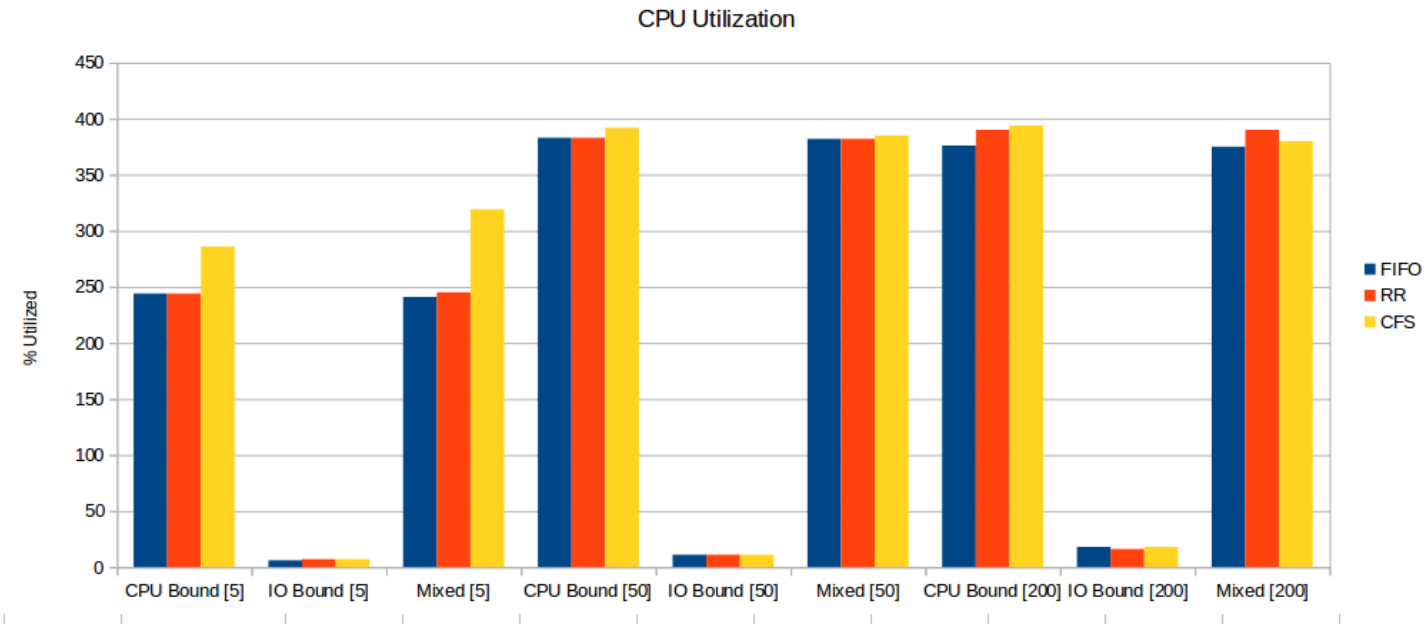
Method

This experiment determines the runtime of the three schedulers a CPU bound, IO bound, and a mixed program. The CPU bound program statistically computes pi over 1000000 iterations. The IO bound copies 102400 bytes from an input file to an output file. The mixed statistically computes pi over 1000000 iterations and writes the result to an output file. Each test is run with each scheduler and with 5, 50 and 200 simultaneous forked processes. The testing machine was natively running Ubuntu 14.10 on an SSD with an Intel i5 3570k, a quad-core CPU with hyper-threading, with 8gb of main memory.

Results


Average Turnaround Time
Time (ms)
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
CPU Bound [5]
IO Bound [5]
Mixed [5]
CPU Bound [50]
IO Bound [50]
Mixed [50]
CPU Bound [200]
IO Bound [200]
Mixed [200]
FIFO
RR
CFS


Number of Premptive Switches
4500
4000
3500
3000
2500
2000
1500
1000
500
0
CPU Bound [5]
IO Bound [5]
Mixed [5]
CPU Bound [50]
IO Bound [50]
Mixed [50]
CPU Bound [200]
IO Bound [200]
Mixed [200]
FIFO
RR
CFS



First In First Out							
	# Processes	Wall	User	System	CPU	i-switched	v-switched
CPU Bound	5	0.09	0.22	0.00	244%	3	13
IO Bound	5	1.02	0.00	0.06	6%	21	1553
Mixed	5	0.09	0.21	0.00	241%	2	12
CPU Bound	50	0.58	2.22	0.00	383%	2	57
IO Bound	50	1.87	0.00	0.20	11%	471	16498
Mixed	50	0.57	2.18	0.00	382%	3	58
CPU Bound	200	2.37	8.92	0.02	376%	16	207
IO Bound	200	3.56	0.02	0.64	18%	706	63797
Mixed	200	2.33	8.73	0.02	375%	14	208

Round Robin							
	# Processes	Wall	User	System	CPU	i-switched	v-switched
CPU Bound	5	0.09	0.22	0.00	244%	2	12
IO Bound	5	1.02	0.00	0.06	7%	18	1981
Mixed	5	0.09	0.21	0.00	245%	2	11
CPU Bound	50	0.58	2.22	0.00	383%	3	58
IO Bound	50	1.81	0.01	0.18	11%	303	15507
Mixed	50	0.57	2.17	0.01	382%	2	57
CPU Bound	200	2.39	8.92	0.02	390%	5	207
IO Bound	200	3.92	0.02	0.64	16%	1276	66643
Mixed	200	2.24	8.73	0.01	390%	6	208

Normal Scheduler							
	# Processes	Wall	User	System	CPU	i-switched	v-switched
CPU Bound	5	0.09	0.26	0.00	286%	98	14
IO Bound	5	0.95	0.00	0.06	7%	45	1946
Mixed	5	0.12	0.38	0.00	319%	193	13
CPU Bound	50	0.57	2.23	0.00	392%	1092	103
IO Bound	50	1.89	0.00	0.22	11%	752	16418
Mixed	50	0.61	2.34	0.01	385%	1098	103
CPU Bound	200	2.26	8.91	0.03	394%	3199	403
IO Bound	200	3.65	0.00	0.66	18%	891	64777
Mixed	200	2.33	8.73	0.02	375%	14	208

Analysis

The fastest turnaround time provides the “best” user experience as programs will respond quickly. Unfortunately this data does not consistently point to one scheduler as being best. In general the CFS scheduler, the default scheduler in Linux, seems to be best for CPU bounded tasks. CFS performs worse in cases with a small amount of simultaneous processes, due to the overhead from more context switches. From this limited set of data preemptive switches appears to increase linearly with number of simultaneous processes.

For systems running primarily or exclusively CPU intensive programs First-in-First-out or Round Robin would be a better scheduling choice than CFS, due to the fact that you don't want these programs getting switched out until they are complete. These schedulers also scale better than CFS and so in systems with a large number of running processes may be better suited.

Conclusion

Although it's hard to definitively say any scheduler is best for normal computer use CFS seems to be better than first-in-first-out or round robin. However in non-typical environments, such as those in server back-ends or super computers FIFO or Round Robin would be more optimal due to their lower overhead.

References

Andy Sayler

Modern Operating Systems

Appendix A – Raw Data

Doing CPUBound 5 times using SCHED_OTHER

wall, user, system, CPU, i-switched, v-switched

0.09 0.26 0.00 286% 98 14

0.07 0.22 0.00 289% 72 14

0.07 0.22 0.00 313% 71 14

0.07 0.22 0.00 301% 82 14

0.07 0.22 0.00 309% 90 14

Doing CPUBound 5 times using SCHED_RR

wall, user, system, CPU, i-switched, v-switched

0.09 0.22 0.00 244% 2 12

0.09 0.21 0.01 246% 3 12

0.09 0.22 0.00 244% 3 13

0.09 0.22 0.00 245% 4 13

0.09 0.22 0.00 247% 3 13

Doing CPUBound 5 times using SCHED_FIFO

wall, user, system, CPU, i-switched, v-switched

0.09 0.22 0.00 244% 3 13

0.09 0.22 0.00 244% 2 12

0.09 0.22 0.00 241% 5 13

0.09 0.22 0.00 247% 4 13

0.09 0.22 0.00 247% 2 12

Doing IOBound 5 times using SCHED_OTHER

wall, user, system, CPU, i-switched, v-switched

0.95 0.00 0.06 7% 45 1946

1.04 0.00 0.06 7% 30 1954

0.99 0.01 0.06 7% 41 1959

0.97 0.00 0.06 7% 40 1970

1.02 0.00 0.07 7% 24 1975

Doing IOBound 5 times using SCHED_RR

wall, user, system, CPU, i-switched, v-switched

1.02 0.00 0.06 7% 18 1981

1.00 0.00 0.05 5% 20 1884

0.97 0.00 0.06 6% 40 1887

1.03 0.00 0.07 7% 22 1967

0.97 0.00 0.07 7% 22 1964

Doing IOBound 5 times using SCHED_FIFO

wall, user, system, CPU, i-switched, v-switched

1.02 0.00 0.06 6% 21 1553

1.04 0.00 0.06 6% 41 1961

1.00 0.00 0.06 6% 32 1967

0.94 0.00 0.06 7% 88 1754

1.05 0.00 0.07 7% 14 1966

Doing Mixed 5 times using SCHED_OTHER

wall, user, system, CPU, i-switched, v-switched

0.12 0.38 0.00 319% 193 13

0.08 0.24 0.00 306% 77 13

0.07 0.21 0.00 306% 76 14

0.07 0.22 0.00 315% 126 14

0.06 0.22 0.00 341% 155 14

Doing Mixed 5 times using SCHED_RR

wall, user, system, CPU, i-switched, v-switched

0.09 0.21 0.00 245% 2 11

0.09 0.21 0.00 245% 2 13

0.08 0.22 0.00 248% 3 12

0.09 0.21 0.00 244% 3 12

0.09 0.22 0.00 245% 3 15

Doing Mixed 5 times using SCHED_FIFO

wall, user, system, CPU, i-switched, v-switched

0.09 0.21 0.00 241% 2 12

0.09 0.22 0.00 246% 4 13

0.09 0.21 0.00 245% 4 12

0.09 0.22 0.00 245% 3 12

0.09 0.21 0.00 245% 2 13

Doing CPUBound 50 times using SCHED_OTHER

wall, user, system, CPU, i-switched, v-switched

0.57 2.23 0.00 392% 1092 103

0.57 2.22 0.01 391% 893 103

0.57 2.23 0.00 390% 762 104

0.57 2.22 0.01 389% 1031 104

0.57 2.23 0.00 390% 781 102

Doing CPUBound 50 times using SCHED_RR

wall, user, system, CPU, i-switched, v-switched

0.58 2.22 0.00 383% 3 58

0.58 2.22 0.00 383% 3 57

0.60 2.22 0.00 366% 7 58

0.58 2.21 0.01 383% 2 57

0.58 2.22 0.00 383% 4 58

Doing CPUBound 50 times using SCHED_FIFO

wall, user, system, CPU, i-switched, v-switched

0.58 2.22 0.00 383% 2 57

0.58 2.22 0.00 383% 3 59

0.58 2.22 0.01 382% 4 58

0.58 2.21 0.02 383% 2 58

0.60 2.22 0.00 368% 7 58

Doing IOBound 50 times using SCHED_OTHER

wall, user, system, CPU, i-switched, v-switched

1.89 0.00 0.22 11% 752 16418

1.91 0.00 0.22 12% 375 16777

1.89 0.00 0.20 10% 444 16823

1.81 0.00 0.21 11% 388 16502

1.88 0.00 0.19 10% 423 17635

Doing IOBound 50 times using SCHED_RR

wall, user, system, CPU, i-switched, v-switched

1.81 0.01 0.18 11% 303 15507

1.93 0.00 0.22 11% 567 16184

1.73 0.00 0.17 10% 411 15286

1.95 0.00 0.20 10% 571 18937

1.84 0.01 0.19 11% 508 16977

Doing IOBound 50 times using SCHED_FIFO

wall, user, system, CPU, i-switched, v-switched

1.87 0.00 0.20 11% 471 16498

1.81 0.00 0.18 10% 319 15399

1.81 0.00 0.21 11% 352 15918

1.84 0.00 0.20 11% 429 17217

1.81 0.00 0.17 10% 451 15811

Doing Mixed 50 times using SCHED_OTHER

wall, user, system, CPU, i-switched, v-switched

0.61 2.34 0.01 385% 1098 103

0.59 2.19 0.00 366% 1002 104

0.58 2.19 0.00 375% 862 104

0.59 2.18 0.01 367% 856 102

0.56 2.18 0.00 386% 857 124

Doing Mixed 50 times using SCHED_RR

wall, user, system, CPU, i-switched, v-switched

0.57 2.17 0.01 382% 2 57

0.59 2.17 0.01 369% 7 58

0.57 2.18 0.00 382% 2 56

0.57 2.18 0.00 382% 2 58

0.57 2.17 0.01 382% 4 57

Doing Mixed 50 times using SCHED_FIFO

wall, user, system, CPU, i-switched, v-switched

0.57 2.18 0.00 382% 3 58

0.59 2.18 0.00 367% 7 58

0.57 2.17 0.01 382% 3 60

0.57 2.17 0.01 383% 2 57

0.57 2.18 0.00 383% 3 57

Doing CPUBound 200 times using SCHED_OTHER

wall, user, system, CPU, i-switched, v-switched

2.26 8.91 0.03 394% 3199 403

2.27 8.90 0.03 393% 3513 407

2.26 8.92 0.00 394% 3257 402

2.26 8.90 0.03 394% 3265 403

2.26 8.92 0.01 394% 3344 399

Doing CPUBound 200 times using SCHED_RR

wall, user, system, CPU, i-switched, v-switched

2.29 8.92 0.02 390% 5 207

2.32 8.91 0.01 383% 11 208

2.38 8.92 0.02 375% 14 207

2.33 8.93 0.02 383% 10 208

2.32 8.91 0.01 383% 11 207

Doing CPUBound 200 times using SCHED_FIFO

wall, user, system, CPU, i-switched, v-switched

2.37 8.92 0.02 376% 16 207

2.33 8.92 0.01 382% 11 207

2.32 8.91 0.01 384% 11 207

2.37 8.91 0.02 375% 13 207

2.32 8.90 0.03 383% 11 207

Doing IOBound 200 times using SCHED_OTHER

wall, user, system, CPU, i-switched, v-switched

3.65 0.00 0.66 18% 891 64777

3.93 0.02 0.62 16% 1615 64562

3.61 0.01 0.57 16% 839 64391

3.91 0.01 0.59 15% 1786 63880

3.65 0.01 0.62 17% 788 65053

Doing IOBound 200 times using SCHED_RR

wall, user, system, CPU, i-switched, v-switched

3.92 0.02 0.64 16% 1456 65063

3.64 0.02 0.62 17% 779 64484

3.58 0.01 0.58 16% 993 64585

3.94 0.01 0.64 16% 1276 66643

3.89 0.01 0.59 15% 1761 64731

Doing IOBound 200 times using SCHED_FIFO

wall, user, system, CPU, i-switched, v-switched

3.56 0.02 0.64 18% 706 63797

3.83 0.02 0.60 16% 1428 64822

3.65 0.00 0.60 16% 642 65493

3.72 0.01 0.60 16% 874 64500

3.84 0.01 0.65 17% 1144 66412

Doing Mixed 200 times using SCHED_OTHER

wall, user, system, CPU, i-switched, v-switched

2.25 8.73 0.03 388% 4019 467

2.25 8.74 0.02 388% 3258 445

2.32 8.75 0.02 378% 3886 595

2.25 8.74 0.02 388% 3571 433

2.25 8.71 0.05 389% 3459 422

Doing Mixed 200 times using SCHED_RR

wall, user, system, CPU, i-switched, v-switched

2.24 8.73 0.01 390% 6 208

2.28 8.73 0.02 383% 11 209

2.33 8.72 0.02 375% 13 209

2.29 8.74 0.01 382% 11 208

2.27 8.73 0.02 384% 11 209

Doing Mixed 200 times using SCHED_FIFO

wall, user, system, CPU, i-switched, v-switched

2.33 8.73 0.02 375% 14 208

2.28 8.71 0.03 383% 10 209

2.28 8.72 0.02 383% 11 207

2.28 8.72 0.03 383% 10 207

2.34 8.76 0.03 374% 14 209

Appendix B – Code

Testscript

```
ITERATIONS=100000000
BYTESTOCOPY=102400
BLOCKSIZE=1024
TIMEFORMAT="wall=%e user=%U system=%S CPU=%P i-switched=%c v-switched=%w"
MAKE="make -s"
NUMPROC="5 50 200"
TESTS="CPUBound IOBound Mixed"
PRIORITIES="SCHED_OTHER SCHED_RR SCHED_FIFO"
echo Building code...
$MAKE clean
$MAKE
echo Starting test runs...
for num in $NUMPROC
do
        for test in $TESTS
        do
                for priority in $PRIORITIES
                do
                        echo Doing $test $num times using $priority
                        echo wall, user, system, CPU, i-switched, v-switched
                        for i in {1..5}
                        do
                           /usr/bin/time -f "$TIMEFORMAT" sudo ./$test $num $priority > /dev/null
                        done
                done
        done
done
```

benchmark.h

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

```
#include <sched.h>
#include <stdio.h>
#include <stdlib.h>     /* atoi */

int forkMe(int numOfFork)
{
  pid_t pid;
  int ii;
  for(ii = 0; ii < numOfFork; ii++)
  {
    pid = fork();
    if(pid == 0)
    {
      break;
    }
  }
  if(pid == 0)
  {
    printf("New Process %d\n", getpid());
    return getpid();
  }
  else
  {
    printf("Parent Process\n");
    while ((pid = waitpid(-1, NULL, 0))) {
            if (errno == ECHILD) {
              break;
            }
    }
    printf("Done with all processes\n");
    return 0;
  }
  return 0;
}
```

CPUBound.c

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

```
#include <sched.h>
#include "benchmark.h"

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

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;
    int processes;
    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;
    iterations = DEFAULT_ITERATIONS;
    /* Set default policy if not supplied */
    if(argc < 3){
            policy = SCHED_OTHER;
    }
    /* Set default number of processes */
    if(argc < 2){
      processes = 1;
    }
    if(argc > 1){
            processes = atol(argv[1]);
            if(iterations < 1){
                fprintf(stderr, "Bad number of processes\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);
        }
}
/* 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));

int pid = forkMe(processes);
if(pid != 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;
}
```

IOBound.c

```
#define _GNU_SOURCE

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

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

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

    int rv;
    int inputFD;
    int outputFD;
    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;
int processes;

transfersize = DEFAULT_TRANSFERSIZE;

blocksize = DEFAULT_BLOCKSIZE;

/* Set default number of processes */
if(argc < 2){
  processes = 1;
}
if(argc > 1){
        processes = atol(argv[1]);
        if(processes < 1){
            fprintf(stderr, "Bad number of processes\n");
            exit(EXIT_FAILURE);
        }
}

/* Set supplied input filename or default if not supplied */

if(strnlen(DEFAULT_INPUTFILENAME, MAXFILENAMELENGTH) >= MAXFILENAMELENGTH){
  fprintf(stderr, "Default input filename too long\n");
  exit(EXIT_FAILURE);
}
strncpy(inputFilename, DEFAULT_INPUTFILENAME, MAXFILENAMELENGTH);
```

```
/* Set supplied output filename base or default if not supplied */
if(strnlen(DEFAULT_OUTPUTFILENAMEBASE, MAXFILENAMELENGTH) >= MAXFILENAMELENGTH){
  fprintf(stderr, "Default output filename base too long\n");
  exit(EXIT_FAILURE);
}
strncpy(outputFilenameBase, DEFAULT_OUTPUTFILENAMEBASE, MAXFILENAMELENGTH);

/* 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);
}

int pid = forkMe(processes);
if(pid != 0){
  /* 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;
}
```

Mixed.c

```
#include <stdio.h>
#include <stdlib.h>     /* atoi */
#include "benchmark.h"

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

#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <fcntl.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>



/* Local Defines */

#define RADIUS (RAND_MAX / 2)

char resultStr[500];

/* 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 pi(long iterations, int fd){
  long i;
  double x, y;
  double inCircle = 0.0;
  double inSquare = 0.0;
```

```
  double pCircle = 0.0;
  double result = 0.0;



  /* 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;
  result = pCircle * 4.0;

  //sprintf(resultStr, "%d\n", result);
  write(fd, resultStr, strlen(resultStr)*sizeof(char));
  return result;
}

int main(int argc, char *argv[])
{
  if(argc < 2)
  {
    perror("Not enough Argument");
    exit(1);
  }
  int processNum = atoi(argv[1]);
  int fd;
  int policy;
  struct sched_param param;
  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);
  }
}
/* 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));
//setScheduler(argv[2]);
int pid = forkMe(processNum);

if(pid == 0)
  {
    return 0;
}

char filename[20];
sprintf(filename, "test%d.txt", pid);
fd = open(filename,O_WRONLY | O_CREAT | O_TRUNC | O_SYNC, S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH);
pi(1000000, fd);
close(fd);
remove( filename );
return 0;
}
```