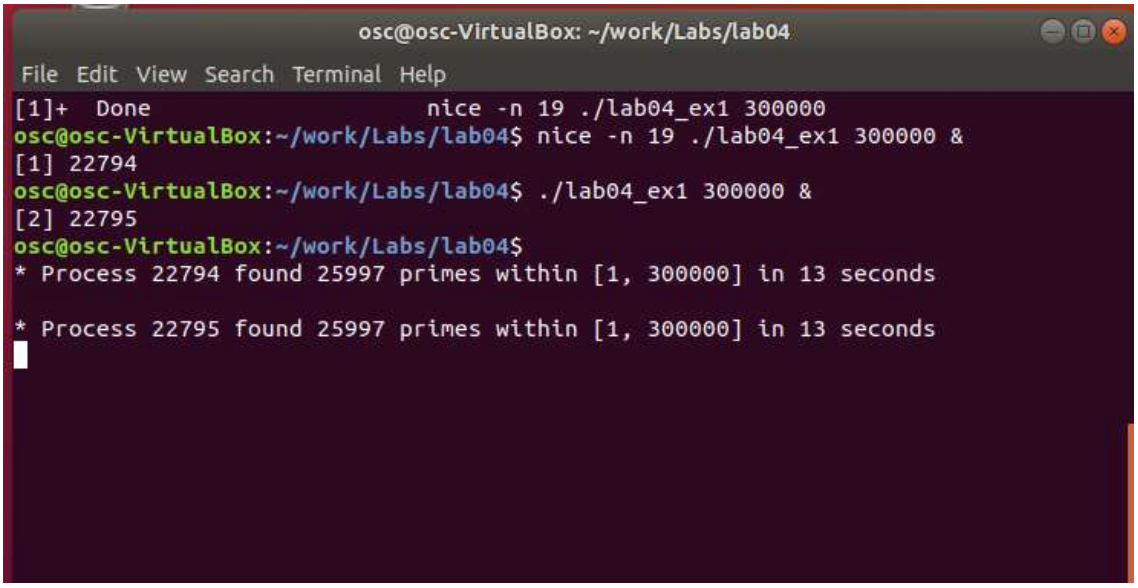
Daniel Funke
CSC 345-01
Lab 04
2/26/2020

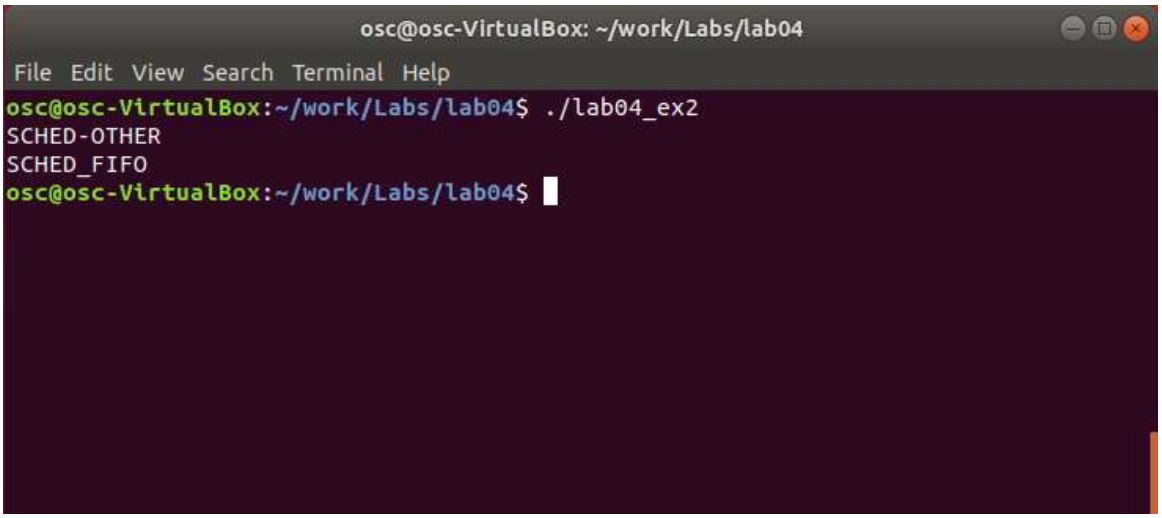
Screenshots



Exercise 1

This exercise runs two processes. The first process is run using the nice command to set its priority to 19, which is lower than the default value of 0. The second process is run at the default priority. Both processes take 13 seconds to complete, and they complete in the order in which they were originally executed. In a less efficient environment the execution of the second process would have trumped the first one and completed in a shorter amount of time.

Daniel Funke
CSC 345-01
Lab 04
2/26/2020



Exercise 2

This program run in this exercise creates five separate threads with the default attribute values. The default scheduling policy is printed to the screen. Next the scheduling policy is set to FIFO instead of OTHER. The new scheduling policy is then printed to the screen.

Daniel Funke
CSC 345-01
Lab 04
2/26/2020

Code for Exercise 1:

```
/** Dan Funke
      CSC345-01
      Lab 4 Exercise 1 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>

int main(int argc, char** argv)
{
      int n = atoi(argv[1]);
      int i, j;
      int count = 0;
      time_t begin = time(NULL);
      pid_t id = getpid();

      for (i = 1; i <= n; ++i) {
            for (j = 2; j < i; ++j) {
                  if (i % j == 0) {
                        break;
                  }
            }
            if (j == i) {
                  //printf("%d ", j);
                  ++count;
            }
      }
```

```
	printf("\n");
	printf("* Process %d found %d primes within [1, %d] in %ld seconds\n",
		id, count, n, time(NULL) - begin);

	return 0;
}
```

Code for Exercise 2:

```
/** Dan Funke
	CSC345-01
	Lab 4 Exercise 2 */

#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS 5

void *runner (void *param);

int main (int argc, char *argv[])
{
	int i, policy;
	pthread_t tid[NUM_THREADS];
	pthread_attr_t attr;

	/* get the default attributes */
	pthread_attr_init(&attr);

	/* get the current scheduling policy */
	if (pthread_attr_getschedpolicy(&attr, &policy) != 0)
		fprintf(stderr, "Unable to get policy.\n");
```

Daniel Funke
CSC 345-01
Lab 04
2/26/2020

```
else {
      if (policy == SCHED_OTHER)
            printf("SCHED-OTHER\n");
      else if (policy == SCHED_RR)
            printf("SCHED_RR\n");
      else if (policy == SCHED_FIFO)
            printf("SCHED_FIFO\n");
}

/* set the scheduling policy - FIFO, RR, or OTHER */
if (pthread_attr_setschedpolicy(&attr, SCHED_FIFO) != 0)
      fprintf(stderr, "Unable to set policy. \n");

/* get new schedule policy */
if (pthread_attr_getschedpolicy(&attr, &policy) != 0)
      fprintf(stderr, "Unable to get policy.\n");
else {
      if (policy == SCHED_OTHER)
            printf("SCHED-OTHER\n");
      else if (policy == SCHED_RR)
            printf("SCHED_RR\n");
      else if (policy == SCHED_FIFO)
            printf("SCHED_FIFO\n");
}

/* create the treads */
for (i = 0; i < NUM_THREADS; i++)
      pthread_create(&tid[i], &attr, runner, NULL);

/* now join on each thread */
```

Daniel Funke
CSC 345-01
Lab 04
2/26/2020

```
	for (i = 0; i < NUM_THREADS; i++)
		pthread_join(tid[i], NULL);
}

/* Each thread will begin control in this function */
void *runner(void *param)
{
	/* do some work... */

	pthread_exit(0);
}
```