Lab 04

Alex Benasutti

CSC345-01

September 25 2019

C Code:

lab04_ex1.c

```
/*
* Alex Benasutti
* CSC345-01
* Lab 4 Exercise 1
*/

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

   // Sieve of Eratosthenes Algorithm

   return 0;
}
```

lab04_ex2.c

```c
/*
* Alex Benasutti
* CSC345-01
* Lab 4 Exercise 2
*/

#include <pthread.h>
#include <stdio.h>

#define NUM_THREADS 5

/* Each thread will begin control in this function */
void *runner(void *param)
{
   /* do some work ... */
   printf("nice\n");

   pthread_exit(0);
}

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

   /* get 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");
   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");

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

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

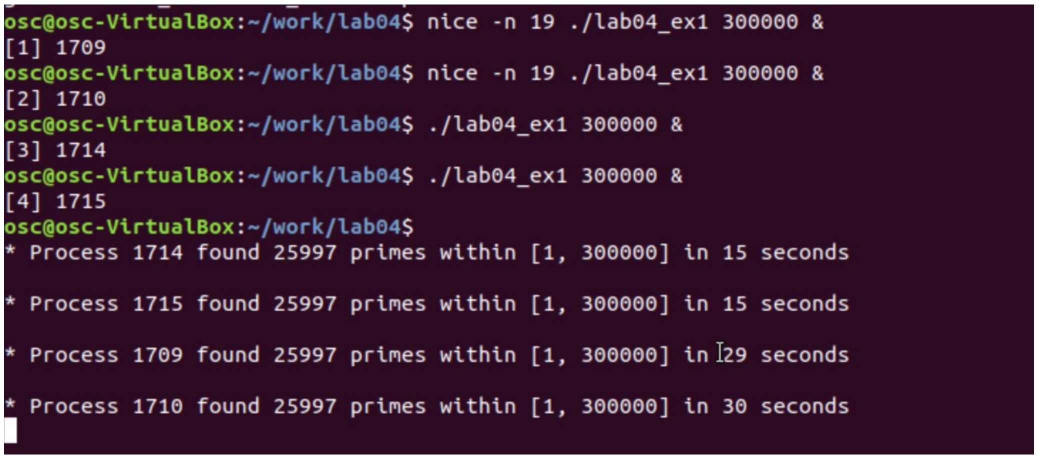
Makefile

```
all:
	gcc -o lab04_ex1 lab04_ex1.c
	gcc -o lab04_ex2 lab04_ex2.c -lpthread
clean:
	rm lab04_ex1
	rm lab04_ex2
```

Results and Analysis

Priority Based Scheduling

The terminal below displays the executing of the example using two different commands - one ran normally and the other using a NICE value of 19. This NICE value gives the latter process a lower priority than the former. As such, the former process is shown to complete first. We run two of each process to keep both processes from running on separate cores, and consequently finishing at the same time.



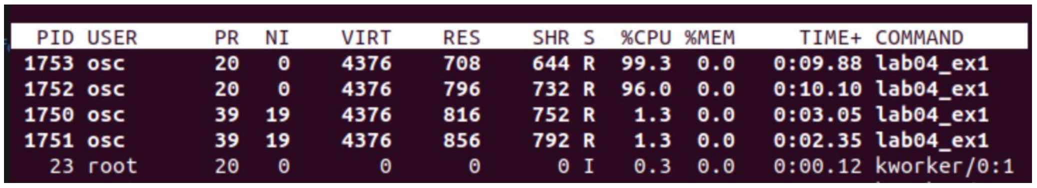
```
osc@osc-VirtualBox:~/work/lab04$ nice -n 19 ./lab04_ex1 300000 &
[1] 1709
osc@osc-VirtualBox:~/work/lab04$ nice -n 19 ./lab04_ex1 300000 &
[2] 1710
osc@osc-VirtualBox:~/work/lab04$ ./lab04_ex1 300000 &
[3] 1714
osc@osc-VirtualBox:~/work/lab04$ ./lab04_ex1 300000 &
[4] 1715
osc@osc-VirtualBox:~/work/lab04$
* Process 1714 found 25997 primes within [1, 300000] in 15 seconds

* Process 1715 found 25997 primes within [1, 300000] in 15 seconds

* Process 1709 found 25997 primes within [1, 300000] in 29 seconds

* Process 1710 found 25997 primes within [1, 300000] in 30 seconds
```

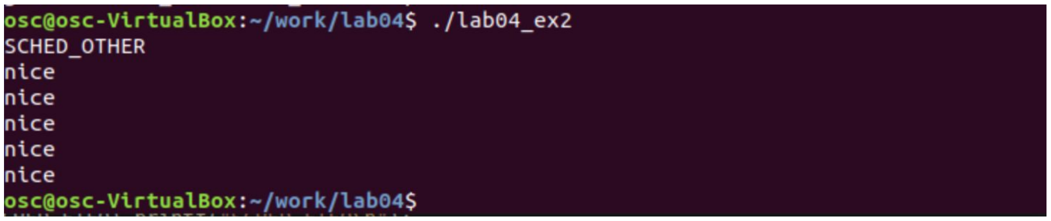
The figure below shows the output of *top* which shows the processes being ran above. The NI value of 19 stands for the NICE value (which is also PR - 20), and one can see that almost no %CPU is being spent on that process, while almost all CPU is spent on the process(es) with higher priority.



```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 1753 osc       20   0    4376    708    644 R  99.3  0.0   0:09.88 lab04_ex1
 1752 osc       20   0    4376    796    732 R  96.0  0.0   0:10.10 lab04_ex1
 1750 osc       39  19    4376    816    752 R   1.3  0.0   0:03.05 lab04_ex1
 1751 osc       39  19    4376    856    792 R   1.3  0.0   0:02.35 lab04_ex1
   23 root      20   0       0      0      0 I   0.3  0.0   0:00.12 kworker/0:1
```

POSIX Real Time Scheduling

The terminal below displays the usage of the POSIX real-time scheduling API example. The code retrieves the scheduling policy, be it round robin (RR), first-in-first-out (FIFO) or other, and executes a predefined number threads using that policy. The runner() procedure is the beginning of control for each thread, which will execute whatever code is contained within, and exit upon completion. For 5 threads, I executed a print statement for the word “nice” plus a new line.



```
osc@osc-VirtualBox:~/work/lab04$ ./lab04_ex2
SCHED_OTHER
nice
nice
nice
nice
nice
osc@osc-VirtualBox:~/work/lab04$
```