CS 460 Scheduling Lab

Shutdown VB and change the System to use only 1 CPU. Use only 1 CPU in VB for this entire exercise. Boot into any working kernel.

Open a terminal

```
sudo pacman -Sy terminator
```

Open terminator and run top (V)

Open terminator and do the following:

```
wget http://zeus.cs.pacificu.edu/chadd/cs460s16/SchedLab.tar.gz

tar xzf SchedLab.tar.gz

cd CS460_SchedulingLab

make
```

This produces a number of executables. We will only use some of these executables today.

PRIORITIES

Try out a few of the executables. Note how much work each executable reports it has done.

```
time ./sleeper 20
```

Work: __________________________

```
time ./CPU 20
```

Work: __________________________

```
time ./IO 20
```

Work: __________________________

Each of the previous executables takes a command line argument that is the runtime in seconds for the process. In the above examples, each process should run for very close to 20 seconds.

Each executable reports the amount of work done and the number of voluntary and involuntary context switches done by that process.

sleeper just continually calls sleep(1) until the runtime is expired.

CPU runs a for loop and does some calculations until the runtime is expired.

IO runs a for loop and prints data to stderr until the runtime is expired.

Open a terminator window. Stretch to fill the screen width.

Right click, Split vertical
On the right half, Right click, Split vertical
On the left half, Right click, Split vertical

Now you have 4 terminals in one window.

In each of the four terminals, type one of the following commands but don't press enter.

```
time ./sleeper 20
```

```
time ./CPU 20
```

```
time ./CPU 20
```

```
time ./IO 20
```

Click the little tiny red/blue/white squares in the top left of any of the 4 terminal windows.

Select Broadcast all.

Press Enter. This will send the enter command to each terminal and run each command at nearly the same time.

Note how much work each executable has done. How does this compare to the original work completed when only one executable is running? Why?

Without typing Enter, in any one of the terminals, type

```
time ./CPU 30
```

This should appear in all four terminals.

Open another terminator window (you should have two single windows and one four way window open).

Press enter in the 4 way window.

In the window running top, look at how much (%) of the CPU each CPU process is getting.

Nice is a command to raise or lower the priority of a process. Only root can raise the priority. You can always lower your priority.

Let's rerun the CPU process in the 4 way window and nice down a process to give that process a lower priority.

Start time ./CPU 30 in all four terminals.

Find the PID of any CPU process in top.

In another terminator window:

```
sudo renice -n 10 -p PID
```

Watch top. Does that PID's share of the CPU drop? Do you see the PR or NI change in top?

Drat!

We have to turn off autogroup for nice to work properly.

```
cat /proc/sys/kernel/sched_autogroup_enabled
su # become root
echo 0 > /proc/sys/kernel/sched_autogroup_enabled # does not stick past reboot!
exit # leave root
```

Now, restart all 4 CPU processes, find one in top and

```
sudo renice -n 10 -p PID
```

Watch top. Does that PID's share of the CPU drop? Do you see the PR or NI change in top?

Description of autogroups.
https://oakbytes.wordpress.com/2012/06/06/linux-scheduler-cfs-and-nice/
http://serverfault.com/questions/405092/nice-level-not-working-on-linux
https://en.wikipedia.org/wiki/Completely_Fair_Scheduler#Fairer_algorithms
http://forum.osdev.org/viewtopic.php?f=15&t=25612

You can also nice your own process at launch.

Click the little tiny red/blue/white squares in the top left of any of the 4 terminal windows.

Select Broadcast off.

Type, but don't hit enter, these commands into

```
time sudo nice -n 0 ./CPU 40
time sudo nice -n -20 ./CPU 40
time sudo nice -n 10 ./CPU 40
time sudo nice -n 19 ./CPU 40
```

Click the little tiny red/blue/white squares in the top left of any of the 4 terminal windows.

Select Broadcast on.

In the window running top:

```
O
COMMAND=CPU
```

How much work did each process get done?

Arrow up in the 4 way window to run these again.

SCHEDULING

Now, let's play with scheduling algorithms.

Linux has a number of scheduling algorithms available:

Real time processes:
SCHED_FIFO

SCHED_RR

Everything else:
SCHED_OTHER
SCHED_BATCH
SCHED_IDLE

Read the man page for sched to understand each algorithm.

Step 0:

schedTest launches 5 threads (via pthreads) and sets the scheduling policy based on the command line argument given. R is RR and F is FIFO. Each thread prints 10 messages containing the thread ID, a progress number, and a time stamp.

At the end, schedTest prints out the number of voluntary and involuntary context switches that occurred.

Note: all output happens just before a process terminates so as to not generate any extra context switches via printf.

```
time sudo ./schedTest R

time sudo ./schedTest F
```

Does it seem like the scheduler is working correctly? Justify your answer.

```
# Bonus: do the following if you have time after completing Step 1.
# Restart with 2 CPUs and run the above tests again. Any difference?
# Restart with 1 CPU and continue.
```

Step 1:

Note: RT scheduling priorities run from 1-99. 99 is highest priority

Note: all output happens just before a process terminates so as to not generate any extra context switches via printf.

schedTestFork launches argv[3] processes (via fork()) and sets the scheduling policy based on argv[1]. R is RR and F is FIFO, B is BATCH, I is IDLE, O is OTHER.

Each thread prints 10 messages containing the thread ID, a progress number, and a time stamp.

argv[2] sets the priority of each process via:
(threadid % argv[2]) + 1 # threadid starts at 0 and increments by 1

As each process ends, the counts of voluntary and involuntary context switches are listed.

Run each of the following command individually.

Because schedTestFork changes its scheduling algorithm, you must run schedTestFork with root privileges.

```
time sudo ./schedTestFork R 1 2

time sudo ./schedTestFork R 98 2

time sudo ./schedTestFork F 1 2
time sudo ./schedTestFork F 98 2
```

Does it seem like the scheduler is working correctly? Justify your answer.

What happens to other processes in VB while these run? Geany? Firefox, etc?

Start in 4 way terminator window to see how processes with different scheduling algorithms interact.

(approx 5 min).

```
time sudo ./schedTestFork R 1 2
time sudo ./schedTestFork R 1 2
time sudo ./schedTestFork R 1 2
time sudo ./schedTestFork R 1 2

time sudo ./schedTestFork R 1 2
time sudo ./schedTestFork R 98 2
time sudo ./schedTestFork R 1 2
time sudo ./schedTestFork O 1 2

time sudo ./schedTestFork R 1 2
time sudo ./schedTestFork R 98 2
time sudo ./schedTestFork R 1 2
time sudo ./schedTestFork I 1 2

time sudo ./schedTestFork R 1 2
time sudo ./schedTestFork R 98 2
time sudo ./schedTestFork F 98 2
time sudo ./schedTestFork I 1 2

time sudo ./schedTestFork R 1 2
time sudo ./schedTestFork R 98 2
time sudo ./schedTestFork F 98 2
time sudo ./schedTestFork B 1 2
```

Does the output match your expectation?