CS 460 Scheduling Lab

Shutdown VB and change the System to use 2 CPUs.

Open terminator!

```
wget http://zeus.cs.pacificu.edu/chadd/cs460s18/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.

Split the terminator window horizontally.

Split the bottom window vertically.

CONFIGURE TOP

In the top window:

```
taskset -c 0 top -u punetid

V
H
f
<arrow down to P = Last Used CPU>
<space>
<arrow up to %CPU>
s
q
s .3
```

RUN THE EXECUTABLES:

In either small window:

Try out a few of the executables. Note how much work each executable reports it has done and how many voluntary and involuntary context switches occur.

```
./sleeper 20

Work: _____  VOL CS:  _____  IVOL CS: _____

./CPU 20

Work: _____  VOL CS:  _____  IVOL CS: _____

./IO 20

Work: _____  VOL CS:  _____  IVOL CS: _____

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.


Group the two smaller windows.

Box menu | New Group ... <enter>

Box menu | select group


Alt-G <send input to all windows in a group>

./CPU 20  # should appear in both small windows


./CPU 20

Work: _____  VOL CS:  _____  IVOL CS: _____


./CPU 20

```

Work: _____ VOL CS: _____ IVOL CS: _____

How does the amount of work compare for the two processes?

How does the amount of work compare against a single ./CPU 20 process?

Let's restrict both processes to the same CPU:

taskset -c 1 ./CPU 20 # should appear in both small windows

taskset -c 1 ./CPU 20

Work: _____ VOL CS: _____ IVOL CS: _____

taskset -c 1 ./CPU 20

Work: _____ VOL CS: _____ IVOL CS: _____

How does the amount of work compare for the two processes?

How does the amount of work compare against two CPU processes running without being restricted to a CPU?

Let's restrict both processes to different CPUs:

Alt-O # Alt-Oh ungroup input

Type in each command below but don't press enter.

Alt-G # regroup input

Press enter

taskset -c 0 ./CPU 20

Work: _____ VOL CS: _____ IVOL CS: _____

taskset -c 1 ./CPU 20

Work: _____ VOL CS: _____ IVOL CS: _____

How does the amount of work compare for the two processes?

How does the amount of work compare against two CPU processes running without being restricted to a CPU?

Let's add a sleeper process to the same CPU:

Alt-O # ungroup input

Type in each command below but don't press enter.

Alt-G # regroup input

Press enter

taskset -c 1 ./CPU 20

Work: _____ VOL CS: _____ IVOL CS: _____

taskset -c 1 ./sleeper 20

Work: _____ VOL CS: _____ IVOL CS: _____

How does the amount of work compare for the two processes?

How does the amount of work compare against two CPU processes running without being restricted to a CPU?

Let's add an IO process to the same CPU:

Alt-O # ungroup input

Type in each command below but don't press enter.

Alt-G # regroup input

Press enter

taskset -c 1 ./CPU 20

Work: _____ VOL CS: _____ IVOL CS: _____

taskset -c 1 ./IO 20

Work: _____ VOL CS: _____ IVOL CS: _____

How does the amount of work compare for the two processes?

How does the amount of work compare against two CPU processes running without being restricted to a CPU?

SCHEDULING

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 pins those threads to core 1 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

```
236
237       time sudo ./schedTest F
238
239       Does it seem like the scheduler is working correctly?
240
241       Justify your answer.
242
243
244
245     Step 1:
246
247       Note: RT scheduling priorities run from 1-99. 99 is highest priority
248
249       Note: all output happens just before a process terminates so as to not
250       generate any extra context switches via printf.
251
252       schedTestFork launches argv[3] processes (via fork()) and pins
253       those threads to core 1 and sets the scheduling policy based on
254       argv[1]. R is RR and F is FIFO, B is BATCH, I is IDLE, O is OTHER.
255
256       Each thread prints 10 messages containing the thread ID, a progress
257       number, and a time stamp.
258
259       argv[2] sets the priority of each process via:
260       (threadid % argv[2]) + 1 # threadid starts at 0 and increments by 1
261
262       As each process ends, the counts of voluntary and involuntary context
263       switches are listed.
264
265       Run each of the following command individually.
266
267       Because schedTestFork changes its scheduling algorithm, you must
268       run schedTestFork with root privileges.
269
270       time sudo ./schedTestFork R 1 2
271
272       time sudo ./schedTestFork R 98 2
273
274       time sudo ./schedTestFork F 1 2
275
276       time sudo ./schedTestFork F 98 2
277
278
279       Does it seem like the scheduler is working correctly?
280       Justify your answer.
281
282
```

Step 1.1:

In a split Terminator window, run the following with grouped input:

```
time sudo ./schedTestFork R 98 2
time ./CPU 20                       work: _____________________________

                                    time (real): ______________________

Just in one terminal alone:
time taskset -c 1 ./CPU 10          work: _____________________________

                                    time (real): ______________________



time sudo ./schedTestFork R 98 10
time taskset -c 1 ./CPU 10          work: _____________________________

                                    time (real): ______________________


time sudo ./schedTestFork R 98 20
time taskset -c 1 ./CPU 10          work: _____________________________

                                    time (real): ______________________


What happens to the work for CPU?

What happens for the real time for CPU?


Start in 2 way terminator window to see how processes with different
scheduling algorithms interact.

The important behavior to watch for is what order the processes
complete in and how the processes interleave their execution.

time sudo ./schedTestFork R 1 5
time sudo ./schedTestFork R 1 5

How do the processes interleave within one execution of schedTestFork?


How do the processes interleave between the two schedTestFork
processes?
```

Do any processes have voluntary context switches?

time sudo ./schedTestFork R 1 5
time sudo ./schedTestFork R 98 5

How do the processes interleave within one execution of schedTestFork?

How do the processes interleave between the two schedTestFork processes?

time sudo ./schedTestFork R 1 5
time sudo ./schedTestFork I 1 5

How do the processes interleave within one execution of schedTestFork?

How do the processes interleave between the two schedTestFork processes?

time sudo ./schedTestFork R 1 5
time sudo ./schedTestFork F 1 5

How do the processes interleave within one execution of schedTestFork?

How do the processes interleave between the two schedTestFork processes?

```
377
378
379
380       time sudo ./schedTestFork R 1 5
381       time sudo ./schedTestFork B 1 5
382
383       How do the processes interleave within one execution of schedTestFork?
384
385
386
387       How do the processes interleave between the two schedTestFork
388       processes?
389
390
```