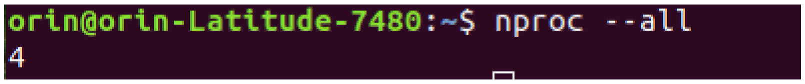
CS 646 – Principles of Operating Systems

Homework 3

Abdur Rouf

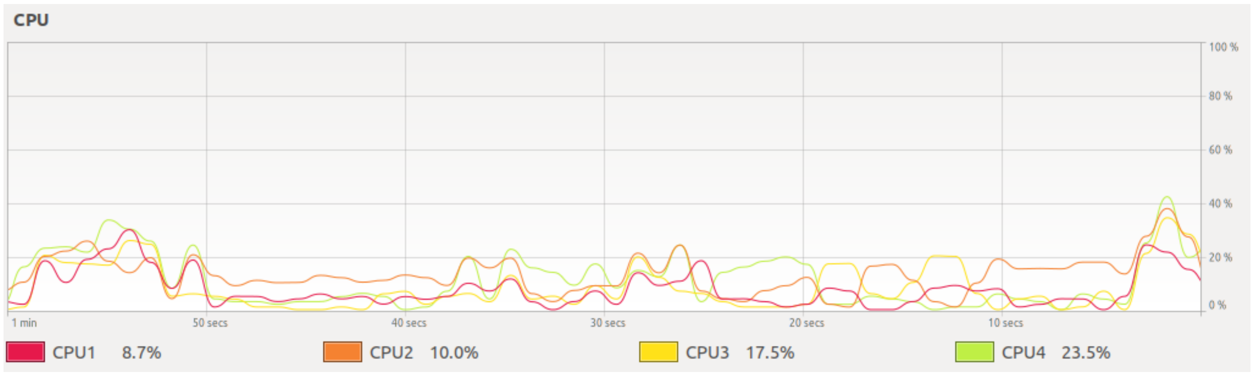
1.

Initially, I check the CPU count by running **nrpoc –all** in my machine. Here, is the output below:

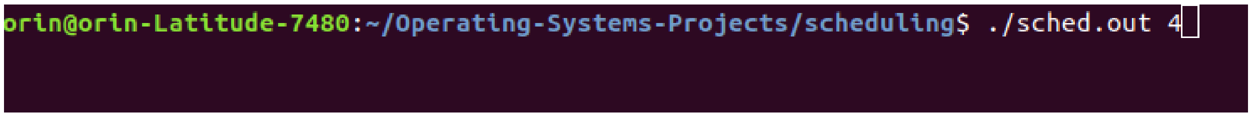


```
orin@orin-Latitude-7480:~$ nproc --all
4
```

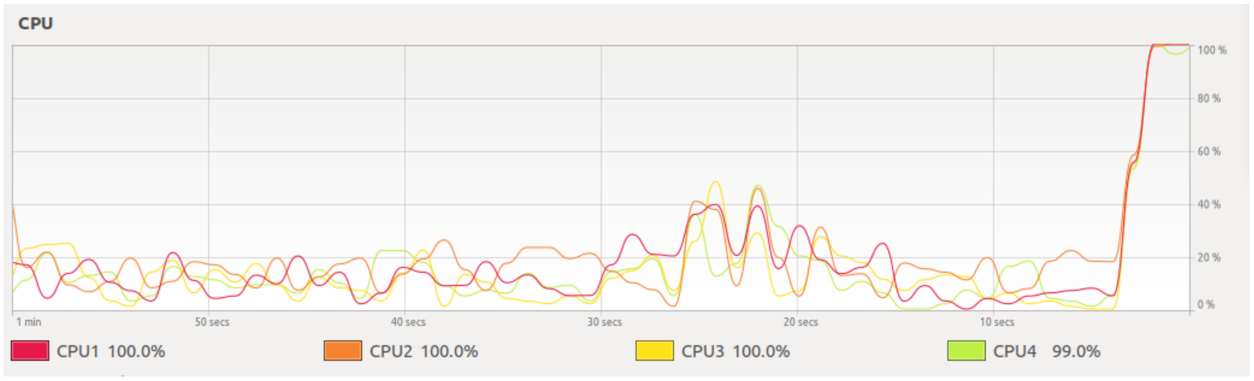
Before running the command ' **./sched.out 4 ',** I took a snapshot of the usage of all CPU cores. It looks like this.



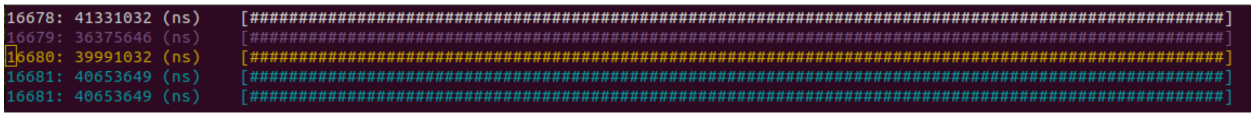
So, according to the figure all cpu cores were being moderately used.
Then, I ran the command **'./sched.out 4'** and the system monitor snapshot looked like this.



```
orin@orin-Latitude-7480:~/Operating-Systems-Projects/scheduling$ ./sched.out 4
```

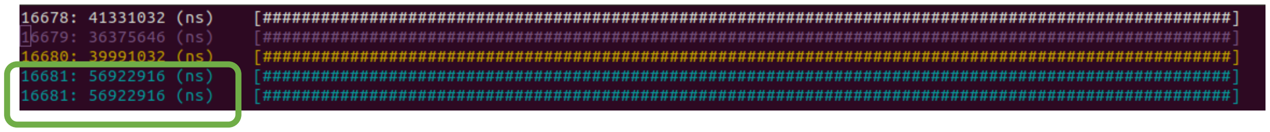


After running the multithreaded program, all CPU cores became busy. Though they are thread, the CPU treated them as a process where each process has process-ID (PID). Since, we're calculating latency for each thread/process and showing the so far maximum latency with the CPU usage, the progress bar looked like this:



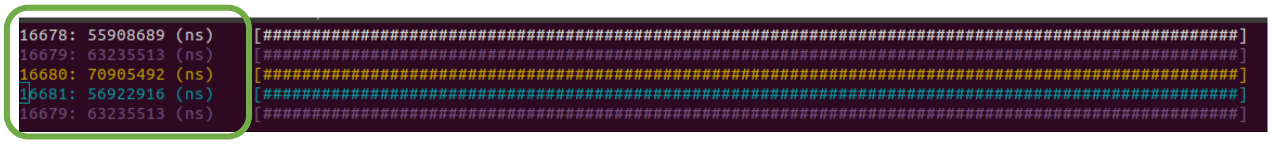
```
16678: 41331032 (ns)    [####################################################################################################]
16679: 36375646 (ns)    [####################################################################################################]
16680: 39991032 (ns)    [####################################################################################################]
16681: 40653649 (ns)    [####################################################################################################]
16681: 40653649 (ns)    [####################################################################################################]
```

Then, I observed the progress bar for a period and found that initially the **max_latency** had changed significantly. After a certain period, this **max_latency** had reached a certain value though, it was changing slowly. Here, I added two more snaps of the progress by observing longer time periods.



```
16678: 41331032 (ns)    [####################################################################################################]
16679: 36375646 (ns)    [####################################################################################################]
16680: 39991032 (ns)    [####################################################################################################]
16681: 56922916 (ns)    [####################################################################################################]
16681: 56922916 (ns)    [####################################################################################################]
```

Here, you have only seen the change of one process **max_latency** though this snapshot had been taken 2-3 minutes later than the first snapshot.



```
16678: 55908689 (ns)    [####################################################################################################]
16679: 63235513 (ns)    [####################################################################################################]
16680: 70905492 (ns)    [####################################################################################################]
16681: 56922916 (ns)    [####################################################################################################]
16679: 63235513 (ns)    [####################################################################################################]
```

Here, all process **max_latency** has been changed.

The main observation of this part experiment is the **max_latency** change. Initially, the **max_latency** was changing swiftly. After a certain time, the change of the **max_latency** converged. However, this convergence was not stable because the **max_latency** was changing very slowly later.

This happened because, initially, in the first iteration, all CPU cores were unoccupied. So, it took a short time to calculate the sum for all processes. Then, latency was increased after all CPU becoming busy state. Additionally, we selected the thread/process count as same CPU. So, the whole latency converged to a certain big value latency. However, the context switching was on going which sometimes held some process for a long time and made change in **max_latency**.

2.

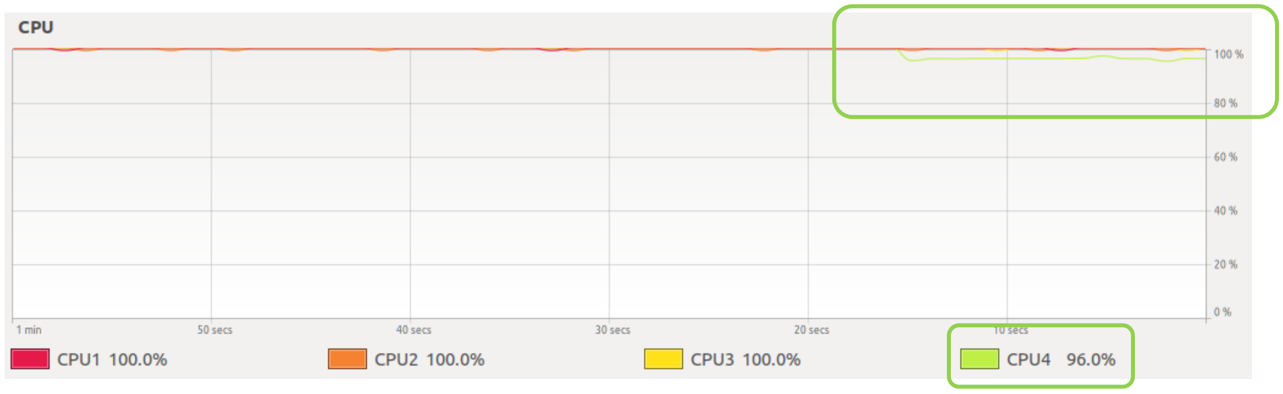
Here, I picked the process with PID: 16678. Then, I opened another terminal and run this command.

```
watch -n.5 grep ctxt /proc/16678/status
```

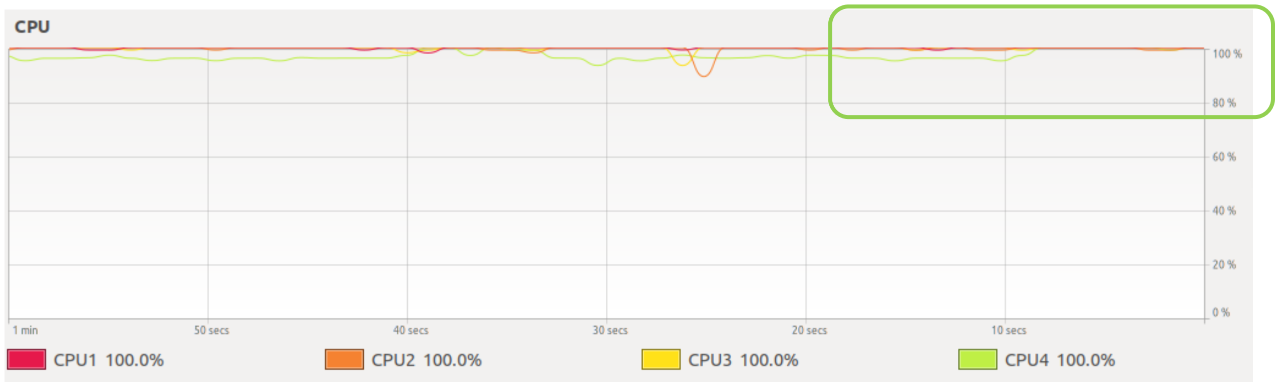
Then, I observed such situations by applying change on above process,

i. Observed the context switching (both voluntary and non-voluntary) for normal scheduling for 1 minute.
ii. Observed the context switching (both voluntary and non-voluntary) for real time scheduling with Round-Robin (RR) for 1 minute.
iii. Observed the context switching (both voluntary and non-voluntary) for normal scheduling for 1 minute **again**.
iv. Observed the context switching (both voluntary and non-voluntary) for real time scheduling with First In First Out (FIFO) for 1 minute.

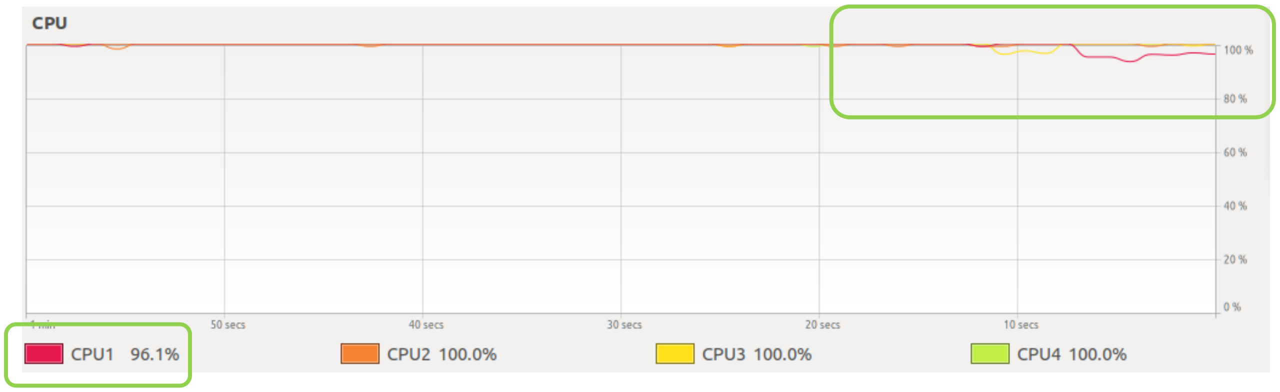
Here are system monitoring snapshots:



This snapshot was taken when scheduling converted from normal to real-time with Round-Robin (RR). And this action made a change in CPU utilization which is illustrated above.



This snapshot reflects the change in CPU utilization after applying scheduling from RR to normal.



This snapshot was taken when scheduling converted from normal to real-time with FIFO. And this action also made a change in CPU utilization which is illustrated above.

I used this command to change the scheduling

```
sudo chrt -p -r 1 16678          # normal to RR
sudo chrt -p -o 0 16678          # RR to normal
sudo chrt -p -f 1 16678          # normal to FIFO
```

In each state, I recorded the number of contexts switched for one minute. Here is the table:

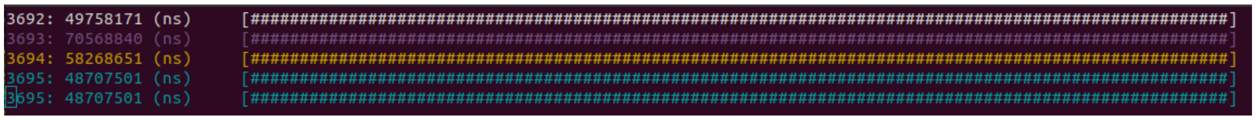
Scheduling Mode	Context Switch	Start state	End state	Total Change
Normal	Voluntary	946	1067	121
	Non-Voluntary	363694	410879	47185
RR	Voluntary	1397	1631	234
	Non-Voluntary	464660	464751	91
Normal	Voluntary	1883	2017	134
	Non-Voluntary	490709	538301	47592
FIFO	Voluntary	2298	2471	173
	Non-Voluntary	577457	577526	69

The main observation from this result is that non voluntary context switch is greater in normal scheduling. When I increased the priority of the process and made it real time scheduling, non-voluntary context switch has been fallen significantly. This is because the process holds the CPU for longer periods which has fallen the non-voluntary context switch count.

Another finding is that, in real time scheduling, one CPU utilization was less than 100%. Here, the observation is, because of the higher priority process, CPU remained unoccupied though it has more job to be done.

3.

Processes progress bar looks like this.



```
3692: 49758171 (ns)     [####################################################################################################]
3693: 70568840 (ns)     [####################################################################################################]
3694: 58268651 (ns)     [####################################################################################################]
3695: 48707501 (ns)     [####################################################################################################]
3695: 48707501 (ns)     [####################################################################################################]
```

In my machine, I've 4 CPU that is why I assigned from CPU – 0 to CPU - 2 to **system cpuset** and CPU-3 to **dedicated cpuset**. Here are the commands I ran to assign CPU to these **cpuset**.

Step 1:

sudo cset set -l

Ran this command to check all the process running on root cpuset

Step 2:

sudo cset set -c 0-2 system

Then ran the create cpuset command with name system and assigned cpu (0-2) to system cpuset.

Step 3:

sudo cset set -c 3 dedicated

Then ran the create cpuset command with name dedicated and assigned cpu 3 to system cpuset.

Step 4:

sudo cset proc -m -f root -t system

sudo cset proc -k -f root -t system

Ran these two commands to transfer all user level and kernel level process from root cpuset to system cpuset.

Step 5:

sudo cset proc -m -p <pid> -t dedicated

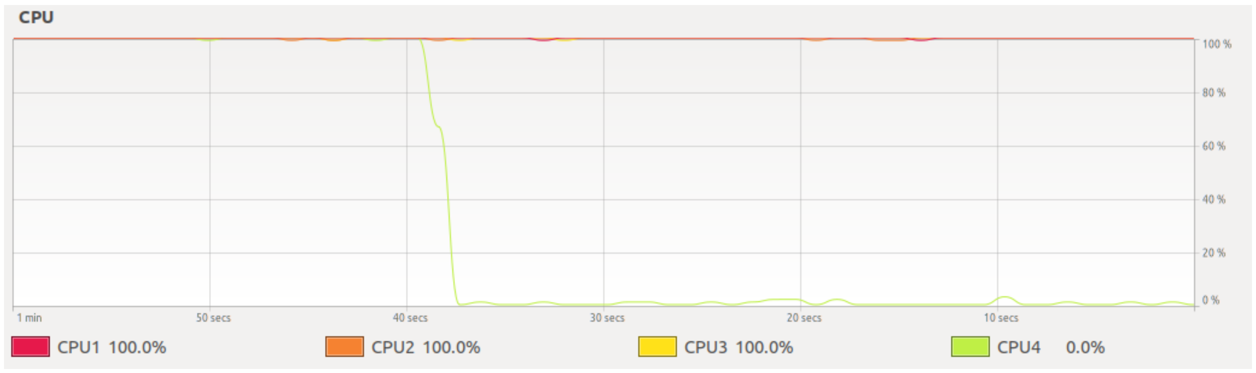
Finally ran this command with process ID equals 3692 which transfer the process with mentioned PID transferred the process from system cpuset to dedicated cpuset.

Here, I observed the context switch before applying dedicated CPU and after applying dedicated CPU to the mentioned process. The table mentioned below:

Dedicated CPU Application	Context Switch	Start state	End state	Total Change
Before	Voluntary	5569	5731	162
	Non-Voluntary	437736	477940	40204
After	Voluntary	3196	3432	236
	Non-Voluntary	323862	336526	12664

Here, before applying dedicated CPU to the process, non-voluntary context switching happened more than after applying dedicated CPU. We know that non-voluntary context switching is a forced process. As other process also shares the same CPU-3 that's why non-voluntary context switch happened more in before state. On the other hand, voluntary context switch happened more in dedicated CPU because the process get more time in CPU which bring the process to termination.

There was another observation from system monitor perspective. When, I transferred all process to CPU 0-2 and made empty to CPU-3, it was clearly visible in system monitor CPU-usage graph. Here is the snapshot of this.



Here, CPU-4 utilization is 0 because at that moment no process had been assigned to this CPU.

4.

Here, I've used the commands that are used in (2). Just process ID different. Here, the dedicated CPU process ID was 3692.
And execute 4 steps like (2)

i. Normal Scheduling to Real Time Scheduling with RR.
ii. Real Time Scheduling RR to Normal Scheduling
iii. Normal Scheduling to Real Time Scheduling with FIFO
iv. Real Time Scheduling FIFO to Normal Scheduling.

I observed each state data for one minute duration. Here is the table:

Scheduling Mode	Context Switch	Start state	End state	Total Change
Normal	Voluntary	3196	3432	236
	Non-Voluntary	323862	336526	12664
RR	Voluntary	3690	3893	203
	Non-Voluntary	350121	350194	73
Normal	Voluntary	4044	4209	165
	Non-Voluntary	354352	366682	12330
FIFO	Voluntary	4802	5007	205
	Non-Voluntary	393843	393916	73

Here, the most interesting finding is, since we assigned a dedicated CPU to the process, the non-voluntary and voluntary context switching behavior will be same regardless of the different real time scheduling policy (RR or FIFO). The data showed the same thing. Normal scheduling to Real time scheduling changing effect already discussed in (2).