



Operating Systems (OS) – Theory

Assignment:02

Process (CPU | I/O) Bound Simulations

Submitted by:

Tayyaba Asif

2019-CE-5

Submitted to:

Sir Waqas Ali

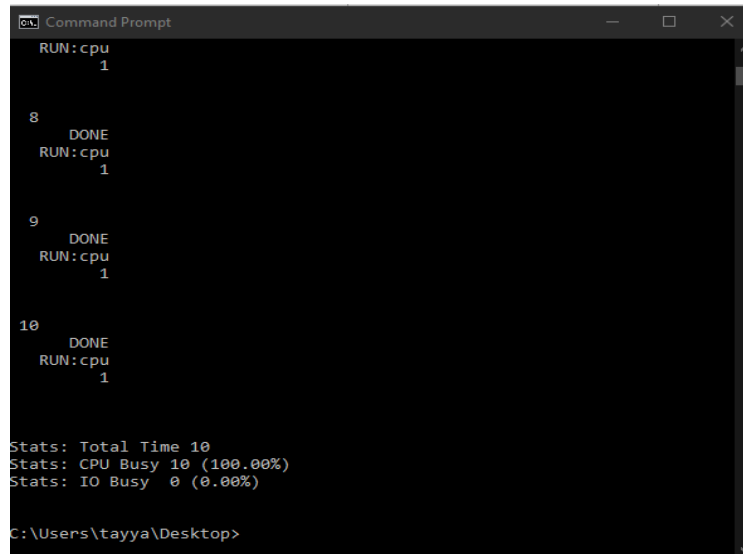
Date: 18th February, 2022

Question No.01:

Run process-run.py with the following flags: -l 5:100,5:100. What should the CPU utilization be (e.g., the percent of time the CPU is in use?) Why do you know this? Use the -c and -p flags to see if you were right.

Answer:

Here all the process are CPU bound so CPU utilization will be 100% since there is no I/O so CPU will not wait.



```
Command Prompt
RUN:cpu
1

8
DONE
RUN:cpu
1

9
DONE
RUN:cpu
1

10
DONE
RUN:cpu
1

Stats: Total Time 10
Stats: CPU Busy 10 (100.00%)
Stats: IO Busy 0 (0.00%)

C:\Users\tayya\Desktop>
```

Question No.02:

Now run with these flags: ./process-run.py -l 4:100,1:0. These flags specify one process with 4 instructions (all to use the CPU), and one that simply issues an I/O and waits for it to be done. How long does it take to complete both processes? Use -c and -p to find out if you were right.

Answer:

Since 4:100 means 4 pure CPU bound instructions and 1:0 means an I/O bound instruction. Instructions will run in the order they are specified. First instruction 4:100 will be completed in 4 clock ticks and then I/O will run and be completed in 5 clock ticks that means both the processes will be completed in 9 clock ticks but scheduler process-run.py is generating the empty tick making the total of 10 ticks.

```
Command Prompt
DONE
WAITING
1
8
DONE
WAITING
1
9
DONE
WAITING
1
10*
DONE
DONE

Stats: Total Time 10
Stats: CPU Busy 5 (50.00%)
Stats: IO Busy 4 (40.00%)

C:\Users\tayya\Desktop>
```

Question No.03:

Switch the order of the processes: -l 1:0,4:100. What happens now? Does switching the order matter? Why? (As always, use -c and -p to see if you were right)

Answer:

Since here first instruction will be I/O bound instruction so both the processes will run in 5 clock ticks because second process can run while CPU is waiting on I/O instruction. But here simulator will add an empty tick at the end making total 6 ticks.

```
Command Prompt
WAITING
RUN:cpu
1
1
4
WAITING
RUN:cpu
1
1
5
WAITING
RUN:cpu
1
1
6*
DONE
DONE

Stats: Total Time 6
Stats: CPU Busy 5 (83.33%)
Stats: IO Busy 4 (66.67%)

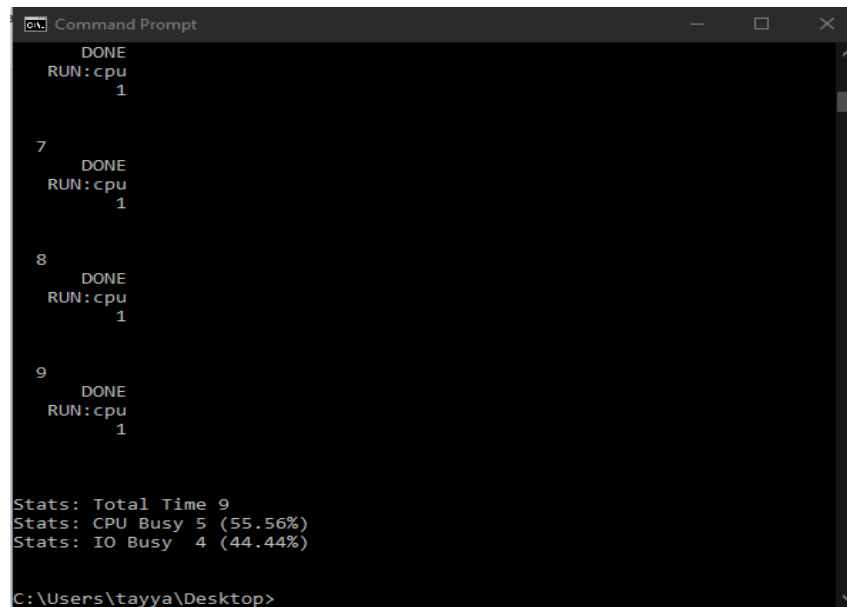
C:\Users\tayya\Desktop>
```

Question No.04:

We'll now explore some of the other flags. One important flag is -S, which determines how the system reacts when a process issues an I/O. With the flag set to SWITCH ON END, the system will NOT switch to another process while one is doing I/O, instead waiting until the process is completely finished. What happens when you run the following two processes (-l 1:0,4:100 -c -S SWITCH ON END), one doing I/O and the other doing CPU work?

Answer:

The resource usage statistics are now almost identical to that in Question No. 2 (9 ticks for both processes to complete), the only difference being that there is no final "empty tick" as the process that does IO runs first.



```
Command Prompt
DONE
RUN:cpu
1

7
DONE
RUN:cpu
1

8
DONE
RUN:cpu
1

9
DONE
RUN:cpu
1

Stats: Total Time 9
Stats: CPU Busy 5 (55.56%)
Stats: IO Busy 4 (44.44%)
C:\Users\tayya\Desktop>
```

Question No.05:

Now, run the same processes, but with the switching behavior set to switch to another process whenever one is WAITING for I/O (-l 1:0,4:100 -c -S SWITCH ON IO). What happens now? Use -c and -p to confirm that you are right.

Answer:

Here answer will be same as of question no.3 because the running I/O instruction first behaves as that of SWITCH_ON_IO.

```
Command Prompt
WAITING
RUN:cpu
1
1

4
WAITING
RUN:cpu
1
1

5
WAITING
RUN:cpu
1
1

6*
DONE
DONE

Stats: Total Time 6
Stats: CPU Busy 5 (83.33%)
Stats: IO Busy 4 (66.67%)

C:\Users\tayya\Desktop>
```

Question No.06:

One other important behavior is what to do when an I/O completes. With -I IO RUN LATER, when an I/O completes, the process that issued it is not necessarily run right away; rather, whatever was running at the time keeps running. What happens when you run this combination of processes? (Run `./process-run.py -l 3:0,5:100,5:100,5:100 -S SWITCH ON IO -I IO RUN LATER -c -p`) Are system resources being effectively utilized?

Answer:

The first process will wait on IO, and the next 3 processes will run to completion, one after the other. Total running time will be 26 ticks with one empty tick at the end. This is not an effective usage of resources as the first process could have been running all the IO operations while the other processes were using CPU.

```
Select Command Prompt
WAITING
DONE
DONE
DONE

1

26
WAITING
DONE
DONE
DONE

1

27*
DONE
DONE
DONE
DONE

Stats: Total Time 27
Stats: CPU Busy 18 (66.67%)
Stats: IO Busy 12 (44.44%)

C:\Users\tayya\Desktop>
```

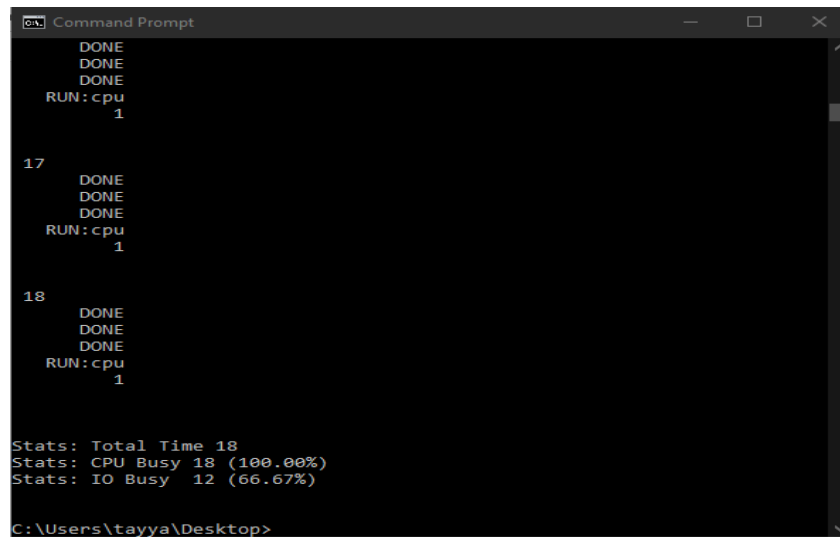
Question No.07:

Now run the same processes, but with -I IO RUN IMMEDIATE set, which immediately runs the process that issued the I/O. How does this behavior differ? Why might running a process that just completed an I/O again be a good idea?

Answer:

In this case the first process will interrupt the other processes in order to start its IO operations, resulting in a reduced computation time of 18 ticks. Resource usage will similarly improve, with CPU usage at 100%.

Running a process that just completed an I/O again is a good idea because that process is more likely to issue another I/O statement than other processes (this is a statistical argument) and in the context of this simulation it is better to kick off I/O operations as soon as possible.



```
Command Prompt
DONE
DONE
DONE
RUN:cpu
1

17
DONE
DONE
DONE
RUN:cpu
1

18
DONE
DONE
DONE
RUN:cpu
1

Stats: Total Time 18
Stats: CPU Busy 18 (100.00%)
Stats: IO Busy 12 (66.67%)
C:\Users\tayya\Desktop>
```

Question No.08:

Now run with some randomly generated processes: -s 1 -l 3:50,3:50 or -s 2 -l 3:50,3:50 or -s 3 -l 3:50,3:50. See if you can predict how the trace will turn out. What happens when you use the flag -I IO RUN IMMEDIATE vs. -I IO RUN LATER? What happens when you use -S SWITCH ON IO vs. -S SWITCH ON END?

Answer:

SWITCH_ON_IO always results in faster run times than SWITCH_ON_END. For the 3 examples given, -I IO_RUN_IMMEDIATE results in exactly the same order of execution as -I IO_RUN_LATER. In all 3 cases, the few possible CPU instructions (each process can have a max of 3 instructions, so each can have a max of 3 CPU instructions) relative to the length of IO operations (5 ticks) means that the processes spend most of their time WAITING on IO, hence there is no opportunity for IO_RUN_IMMEDIATE to evict a currently running process.

```
Command Prompt
RUN:cpu
WAITING
1
1
8*
DONE
RUN:io
1
9
DONE
WAITING
1
10
DONE
WAITING
1
11
DONE
WAITING
1
12
DONE
WAITING
1
13*
DONE
RUN:cpu
1
Stats: Total Time 13
Stats: CPU Busy 6 (46.15%)
Stats: IO Busy 9 (69.23%)

Command Prompt
1
13*
DONE
RUN:io
1
14
DONE
WAITING
1
15
DONE
WAITING
1
16
DONE
WAITING
1
17
DONE
WAITING
1
18*
DONE
RUN:cpu
1
Stats: Total Time 18
Stats: CPU Busy 6 (33.33%)
Stats: IO Busy 12 (66.67%)
C:\Users\tayya\Desktop>
```