# 【OS】Day7(3)

## 【Ch2】Process Homework(Simulation)

*Exercise 1*

1. 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.

The CPU executes process0 first. When it finishes executing process0, it will start to execute process1. The total time cost is 10 time units.

```
PS D:\ostep-homework\cpu-intro> python .\process-run.py -l 5:100,5:100 -c -p
Time        PID: 0        PID: 1           CPU          IOs
  1        RUN:cpu         READY             1
  2        RUN:cpu         READY             1
  3        RUN:cpu         READY             1
  4        RUN:cpu         READY             1
  5        RUN:cpu         READY             1
  6           DONE       RUN:cpu             1
  7           DONE       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%)
```

*Exercise 2*

2. 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.

CPU will execute process0 first, which has 4 CPU instructions and cost 4 time units.

After finishing process0, it will execute process1, which have one I/O instruction(cost 2 time units) and keep the I/O busy for 5 time units.

Thus, the execution of the two processes will cost 11 time units.

```
PS D:\ostep-homework\cpu-intro> python .\process-run.py -l 4:100,1:0 -c -p
Time        PID: 0        PID: 1           CPU           IOs
  1        RUN:cpu         READY             1
  2        RUN:cpu         READY             1
  3        RUN:cpu         READY             1
  4        RUN:cpu         READY             1
  5           DONE        RUN:io             1
  6           DONE       WAITING                           1
  7           DONE       WAITING                           1
  8           DONE       WAITING                           1
  9           DONE       WAITING                           1
 10           DONE       WAITING                           1
 11*          DONE   RUN:io_done             1

Stats: Total Time 11
Stats: CPU Busy 6 (54.55%)
Stats: IO Busy  5 (45.45%)
```

*Exercise 3*

3. 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)

*Switch order matters.*

In this case, the I/O instruction is executed first. CPU can run process1 when process0 is waiting for I/O to be done.

After process1 is finished, there is still 1 time unit left for the I/O to do its job and 1 time unit cost for the CPU to end the I/O.

Thus, the total time cost is 7 time units.

*Exercise 4*

4. 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?

With the `-S` flag set to `SWITCH_ON_END`, the CPU is not doing work when the I/O instruction is being executed.

```
PS C:\ostep-homework\cpu-intro> python .\process-run.py -l 1:0,4:100 -c -S SWITCH_ON_END
Time          PID: 0          PID: 1            CPU           IOs
  1          RUN:io           READY             1
  2         WAITING           READY                             1
  3         WAITING           READY                             1
  4         WAITING           READY                             1
  5         WAITING           READY                             1
  6         WAITING           READY                             1
  7*    RUN:io_done           READY             1
  8            DONE         RUN:cpu             1
  9            DONE         RUN:cpu             1
 10            DONE         RUN:cpu             1
 11            DONE         RUN:cpu             1
```

*Exercise 5*

5. 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.

Now, process1(the 4 CPU instructions) will be executed while waiting for I/O. T

he I/O instruction executes for 1 time unit, then passes the control to process1 and have it execute for 4 time units.

Then, another time unit is spent for I/O. Finally, the control was passed to CPU when the I/O is done.

*Exercise 6*

6. 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?

The system resources are not being effectively utilized. After the first I/O finishes, instead of executing the second I/O instruction and then the second CPU instruction, the second CPU instruction was executed.

This caused a waste of time later. During the second I/O instruction is being executed, the CPU just waits without doing any work.

```
PS C:\ostep-homework\cpu-intro> python .\process-run.py -l 3:0,5:100,5:100,5:100 -S SWITCH_ON_IO -I IO_RUN_LATER -c -p
Time        PID: 0        PID: 1        PID: 2        PID: 3           CPU           IOs
  1         RUN:io         READY         READY         READY             1
  2        WAITING       RUN:cpu         READY         READY             1             1
  3        WAITING       RUN:cpu         READY         READY             1             1
  4        WAITING       RUN:cpu         READY         READY             1             1
  5        WAITING       RUN:cpu         READY         READY             1             1
  6        WAITING       RUN:cpu         READY         READY             1             1
  7*         READY          DONE       RUN:cpu         READY             1
  8          READY          DONE       RUN:cpu         READY             1
  9          READY          DONE       RUN:cpu         READY             1
 10          READY          DONE       RUN:cpu         READY             1
 11          READY          DONE       RUN:cpu         READY             1
 12          READY          DONE          DONE       RUN:cpu             1
 13          READY          DONE          DONE       RUN:cpu             1
 14          READY          DONE          DONE       RUN:cpu             1
 15          READY          DONE          DONE       RUN:cpu             1
 16          READY          DONE          DONE       RUN:cpu             1
 17    RUN:io_done          DONE          DONE          DONE             1
 18         RUN:io          DONE          DONE          DONE             1
 19        WAITING          DONE          DONE          DONE                           1
 20        WAITING          DONE          DONE          DONE                           1
 21        WAITING          DONE          DONE          DONE                           1
 22        WAITING          DONE          DONE          DONE                           1
 23        WAITING          DONE          DONE          DONE                           1
 24*   RUN:io_done          DONE          DONE          DONE             1
 25         RUN:io          DONE          DONE          DONE             1
 26        WAITING          DONE          DONE          DONE                           1
 27        WAITING          DONE          DONE          DONE                           1
 28        WAITING          DONE          DONE          DONE                           1
 29        WAITING          DONE          DONE          DONE                           1
 30        WAITING          DONE          DONE          DONE                           1
 31*   RUN:io_done          DONE          DONE          DONE             1

Stats: Total Time 31
Stats: CPU Busy 21 (67.74%)
Stats: IO Busy  15 (48.39%)
```

## *Exercise 7*

7. 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?

Now, the CPU is fully utilized.

```
PS C:\ostep-homework\cpu-intro> python .\process-run.py -l 3:0,5:100,5:100,5:100 -S SWITCH_ON_IO -I IO_RUN_IMMEDIATE -c
-p
Time        PID: 0       PID: 1       PID: 2       PID: 3          CPU          IOs
  1         RUN:io        READY        READY        READY            1
  2        WAITING      RUN:cpu        READY        READY            1            1
  3        WAITING      RUN:cpu        READY        READY            1            1
  4        WAITING      RUN:cpu        READY        READY            1            1
  5        WAITING      RUN:cpu        READY        READY            1            1
  6        WAITING      RUN:cpu        READY        READY            1            1
  7*   RUN:io_done         DONE        READY        READY            1
  8         RUN:io         DONE        READY        READY            1
  9        WAITING         DONE      RUN:cpu        READY            1            1
 10        WAITING         DONE      RUN:cpu        READY            1            1
 11        WAITING         DONE      RUN:cpu        READY            1            1
 12        WAITING         DONE      RUN:cpu        READY            1            1
 13        WAITING         DONE      RUN:cpu        READY            1            1
 14*   RUN:io_done         DONE         DONE        READY            1
 15         RUN:io         DONE         DONE        READY            1
 16        WAITING         DONE         DONE      RUN:cpu            1            1
 17        WAITING         DONE         DONE      RUN:cpu            1            1
 18        WAITING         DONE         DONE      RUN:cpu            1            1
 19        WAITING         DONE         DONE      RUN:cpu            1            1
 20        WAITING         DONE         DONE      RUN:cpu            1            1
 21*   RUN:io_done         DONE         DONE         DONE            1

Stats: Total Time 21
Stats: CPU Busy 21 (100.00%)
Stats: IO Busy  15 (71.43%)
```

*Exercise 8*

8. 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`?

When `IO_RUN_IMMEDIATE` is set, the CPU will run another I/O instruction right after the current instruction finishes, and thus fully utilizes the time waiting for the data to be fetched.

When `SWITCH_ON_IO` is set, the CPU transfers control to another process when an I/O instruction is invoked.