**The Process**

**Submission:**

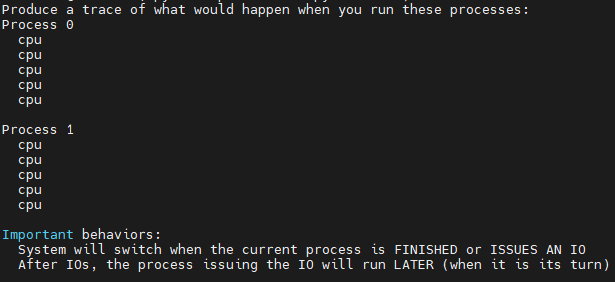
• Deadline: Wednesday, September 20, 2023, 8:00 pm HKT.

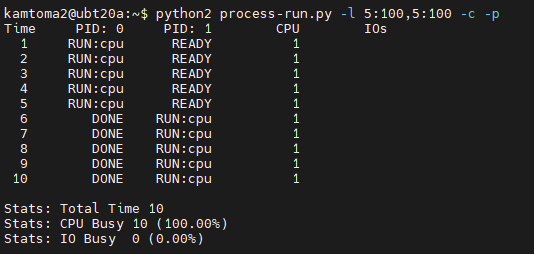
• Submit this answer sheet via Canvas->Assignments->Tutorials->Tutorial 2.

**Questions**

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.

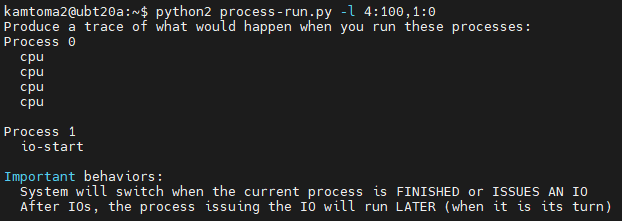
**Answer: 100%. I know this by the flags: 5:100, since the 5 represents as there is 5 CPU process and 100 represent that with 100 percentage of usage.**

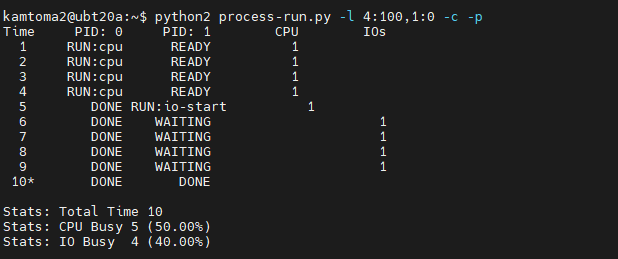
****

****

1. Now run with these ﬂags: -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.

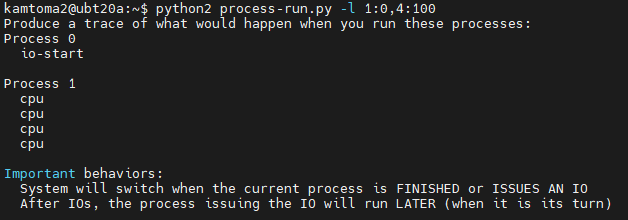
**Answer: Total time will be 10. It is because after 4:100 flag, which means there will be 4 CPU process and it takes total time 4. Then, there is 1:0 flag, which means there will be an I/O be issued and it takes total time 4 as well. When each I/O is issued, the process moves to a WAITING state, and while the device is busy servicing the I/O, the CPU is idle. The I/O operation in the second process would be subject to I/O latency, which would further increase the total time. Therefore, after the initialization, there will be 1 time for both CPU and I/O to change the state to DONE.**

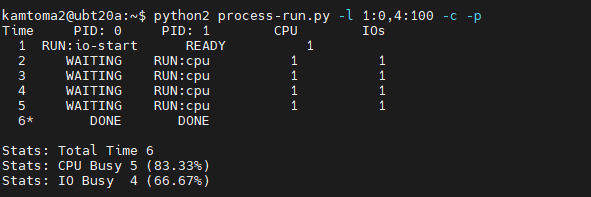
****

****

1. 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)

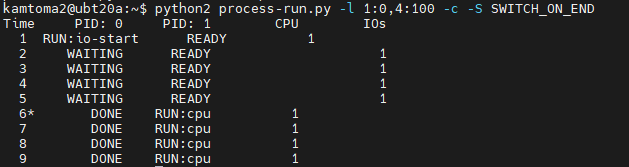
**Answer: Switching the order does matter. The time will either be longer or shorter due to the processing procedure. From this flags: 1:0,4:100, We can see I/O will start RUN first, then the CPU processes follow. Therefore, when I/O is issued and finished first. Thus, CPU will not be idled in the instructions after. Hence, the time will shorter than the flags we used in Question 2.**

****

****

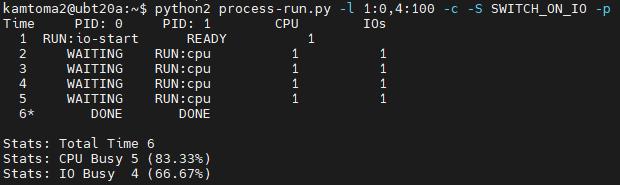
1. We’ll now explore some of the other ﬂags. One important ﬂag is -S, which determines how the system reacts when a process issues an I/O. With the ﬂag set to SWITCH\_ON\_END, the system will **NOT** switch to another process while one is doing I/O, instead waiting until the process is completely ﬁnished. 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 -S SWITCH\_ON\_END flag means that the system will not switch to another process while one is doing I/O. Instead, it will wait until the process is completely finished, which means the total time will be longer, since the CPU has to wait 4 time to run the processes.**

****

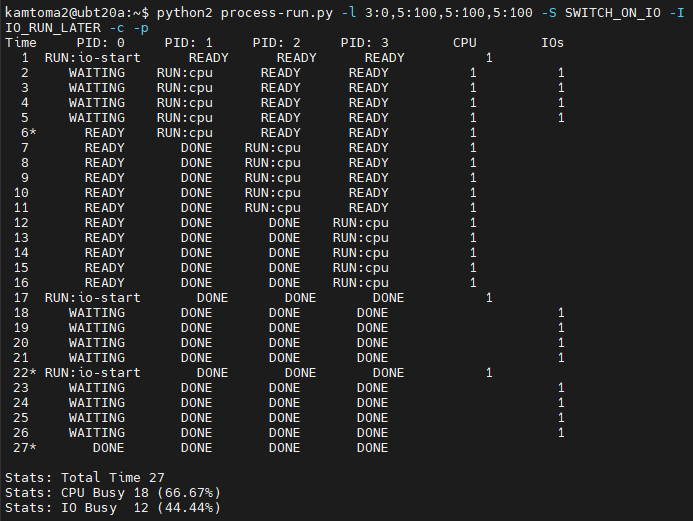
1. Now, run the same processes, but with the switching behavior set to switch to another process whenever one is **WAITING** for I/O (-l 1:0,4:100 -c -S SWITCH\_ON\_IO). What happens now? Use -c and -p to conﬁrm that you are right.

**Answer: It should be same to the result of Question 3. Since SWITCH\_ON\_IO means that the system will switch to the other processes while one is doing I/O.**

****

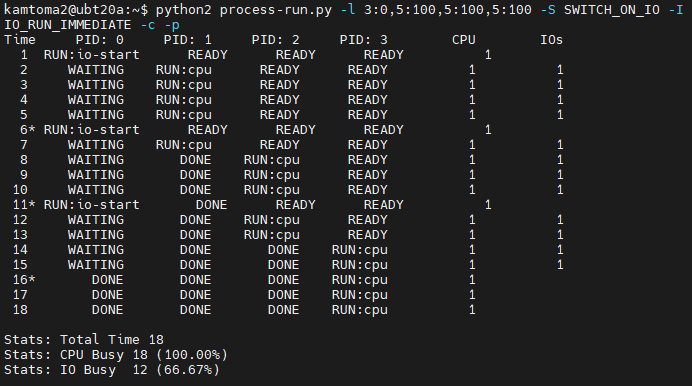
1. One other important behavior is what to do when an I/O completes. With -I IO\_RUN\_LATER, when an I/O completes, the process that issued it is not necessarily run right away; rather, whatever was running at the time keeps running. What happens when you run this combination of processes? (-l 3:0,5:100,5:100,5:100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

**Answer: -I IO\_RUN\_LATER means that the I/O will be run later after all CPU processes are all completely finished. Therefore, only the first one will start, but the remaining two I/O will be issued after the state of 5 + 5 + 5 = 15 CPU processes changed to READY. Hence, it is not being effectively utilized.**

****

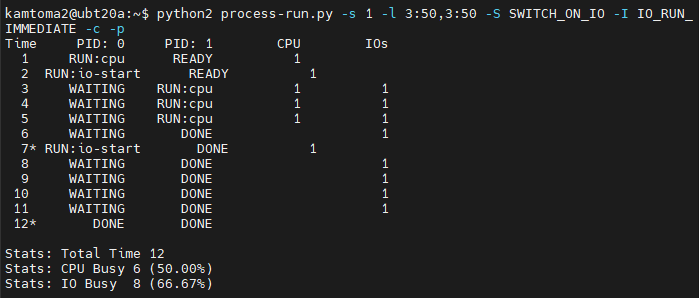
1. Now run the same processes, but with -I IO\_RUN\_IMMEDIATE set, which immediately runs the process that issued the I/O. How does this behavior differ? Why might running a process that just completed an I/O again be a good idea?

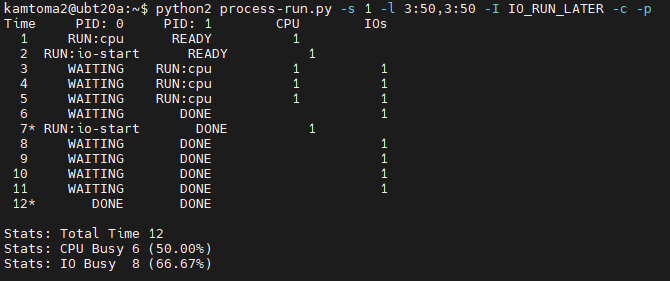
**Answer: -I IO\_RUN\_IMMEDIATE indicated that the I/O and CPU can run at the same time, therefore no time is wasted in waiting for all CPU to be done, as Question 6. Therefore, it is a good idea that let I/O to run immediate.**

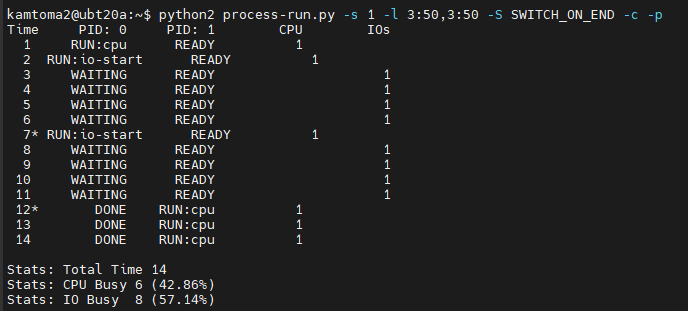
****

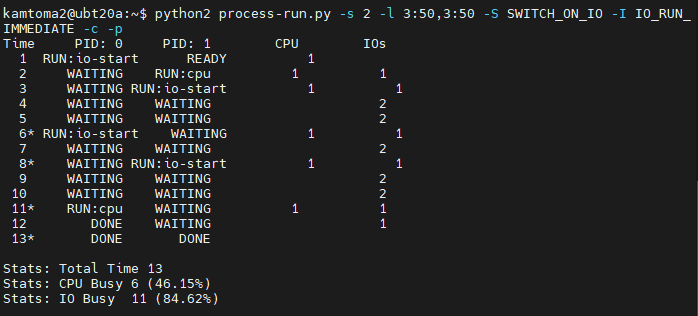
1. Now run with some randomly generated processes: -s 1 -l 3:50,3:50 or -s 2 -l 3:50,3:50 or -s 3 -l 3:50,3:50. See if you can predict how the trace will turn out. What happens when you use the ﬂag -I IO\_RUN\_IMMEDIATE vs. -I IO\_RUN\_LATER? What happens when you use -S SWITCH\_ON\_IO vs. -S SWITCH\_ON\_END?

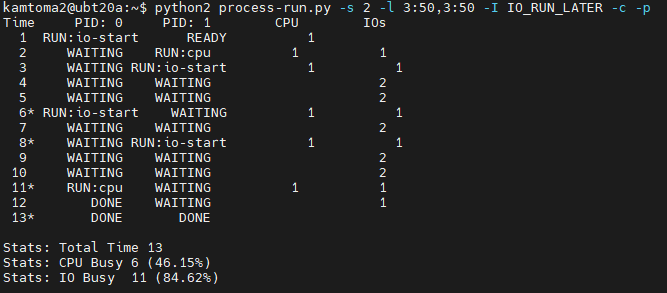
**Answer: As we can know that if using flag -S SWITCH\_ON\_END will have a longer time, since only either CPU or I/O work at the time. Then, -I IO\_RUN\_LATER may also have a longer time if there is more than one I/O issued.**

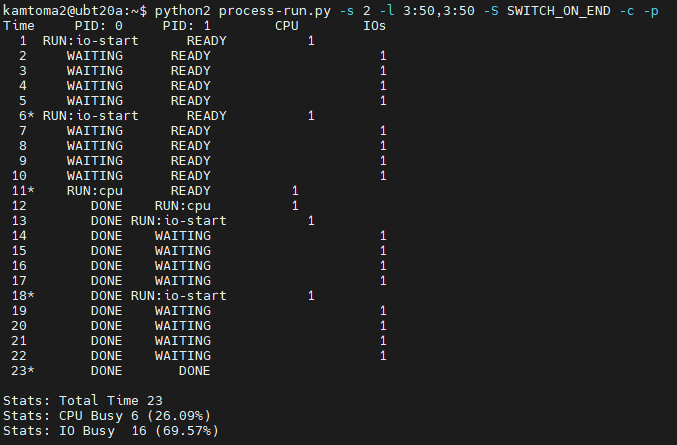
****

****

****

****

****

****

