Operating Systems Assignment 2 (CLO 1, PLO1, Total Marks 40)

This program, process-run.py, allows you to see how process states change as programs run and either use the CPU (e.g., perform an add instruction) or do I/O (e.g., send a request to a disk and wait for it to complete).

1. [4] Run process-run.py with the following flags: -l 8:100,2:100. 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.

Run process-run.py with the following flags: -l 8:100,2:100

```
(base) sneaky@sneaky-Lenovo-ideapad-520-15IKB:~/Documents/c/005_HA2/ostep-homework/cpu-intro$ ./process-run.py -l 8:100,2:100
Produce a trace of what would happen when you run these processes:
Process 0
  cpu
  cpu
  cpu
  cpu
  cpu
  cpu
  cpu
  cpu

Process 1
  cpu
  cpu

Important behaviors:
  System will switch when the current process is FINISHED or ISSUES AN IO
  After IOs, the process issuing the IO will run LATER (when it is its turn)
```

What should the CPU utilization be (e.g., the percent of time the CPU is in use?)
CPU utilization should be 100% or both of the processes.

Why do you know this?
I know this because in the listing processes, we mentioned that both processes are 100% likely to use cpu or not to initiate I/O requests.

```
(base) sneaky@sneaky-Lenovo-ideapad-520-15IKB:~/Documents/c/005_HA2/ostep-homework/cpu-intro$ ./process-run.py -h
Usage: process-run.py [options]

Options:
  -h, --help            show this help message and exit
  -s SEED, --seed=SEED  the random seed
  -P PROGRAM, --program=PROGRAM
                        more specific controls over programs
  -l PROCESS_LIST, --processlist=PROCESS_LIST
                        a comma-separated list of processes to run, in the
                        form X1:Y1,X2:Y2,... where X is the number of
                        instructions that process should run, and Y the
                        chances (from 0 to 100) that an instruction will use
                        the CPU or issue an IO (i.e., if Y is 100, a process
                        will ONLY use the CPU and issue no I/Os; if Y is 0, a
                        process will only issue I/Os)
```

Use the -c and -p flags to see if you were right.
Yes! It appears correct to me.

```
(base) sneaky@sneaky-Lenovo-ideapad-520-15IKB:~/Documents/c/005_HA2/ostep-homework/cpu-intro$ ./process-run.py -l 8:100,2:100 -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        RUN:cpu         READY             1
  7        RUN:cpu         READY             1
  8        RUN:cpu         READY             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%)
```

2. [4] Now run with these flags: ./process-run.py -l 6:100,1:0. These flags specify one process with 6 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.

   How long does it take to complete both processes? Use -c and -p to find out if you were right.

   It should take 8 time units to complete both processes(6 units for process 0, and 2 time units for process 1). But in reality it is taking 13 time units.

```
sneaky@sneaky-Lenovo-ideapad-520-15IKB:~/Documents/c/005_HA2/ostep-homework/cpu-intro$ ./process-run.py -l 6:100,1:0 -c -p
     PID: 0        PID: 1           CPU           IOs
    RUN:cpu         READY             1
    RUN:cpu         READY             1
    RUN:cpu         READY             1
    RUN:cpu         READY             1
    RUN:cpu         READY             1
    RUN:cpu         READY             1
       DONE        RUN:io             1
       DONE       BLOCKED                           1
       DONE       BLOCKED                           1
       DONE       BLOCKED                           1
       DONE       BLOCKED                           1
       DONE       BLOCKED                           1
       DONE   RUN:io_done             1

Total Time 13
CPU Busy 8 (61.54%)
IO Busy  5 (38.46%)
```

3. [4] Switch the order of the processes: -l 1:0,6:100. What happens now? Does switching the order matter? Why? (As always, use -c and -p to see if you were right)

   What happens now? Does switching the order matter? Why?
   Yes! Order matters as apparently it is taking 8 time units compared to 13 time units which it was taking earlier. Because in current order instructions are pipelined better like during process zero is in a blocked state, It is catering process one.

```
(base) sneaky@sneaky-Lenovo-ideapad-520-15IKB:~/Documents/c/005_HA2/ostep-homework/cpu-intro$ ./process-run.py -l 1:0,6:100 -c -p
Time        PID: 0        PID: 1           CPU           IOs
  1         RUN:io         READY             1
  2        BLOCKED       RUN:cpu             1             1
  3        BLOCKED       RUN:cpu             1             1
  4        BLOCKED       RUN:cpu             1             1
  5        BLOCKED       RUN:cpu             1             1
  6        BLOCKED       RUN:cpu             1             1
  7*         READY       RUN:cpu             1
  8    RUN:io_done          DONE             1

Stats: Total Time 8
Stats: CPU Busy 8 (100.00%)
Stats: IO Busy  5 (62.50%)
```

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 2:0,8:100 -c -S SWITCH ON END), one doing I/O and the other doing CPU work?

   When I run with SWITCH_ON_END, It effectively means that IO blocked state time would not be leveraged by the OS, and the CPU would stay idle during that time.

```
(base) sneaky@sneaky-Lenovo-ideapad-520-15IKB:~/Documents/c/005_HA2/ostep-homework/cpu-intro$ ./process-run.py -l 2:0,8:100 -c -S SWITCH_ON_END -c -p
Time        PID: 0        PID: 1           CPU           IOs
  1         RUN:io         READY             1
  2        BLOCKED         READY                           1
  3        BLOCKED         READY                           1
  4        BLOCKED         READY                           1
  5        BLOCKED         READY                           1
  6        BLOCKED         READY                           1
  7*   RUN:io_done         READY             1
  8         RUN:io         READY             1
  9        BLOCKED         READY                           1
 10        BLOCKED         READY                           1
 11        BLOCKED         READY                           1
 12        BLOCKED         READY                           1
 13        BLOCKED         READY                           1
 14*   RUN:io_done         READY             1
 15           DONE       RUN:cpu             1
 16           DONE       RUN:cpu             1
 17           DONE       RUN:cpu             1
 18           DONE       RUN:cpu             1
 19           DONE       RUN:cpu             1
 20           DONE       RUN:cpu             1
 21           DONE       RUN:cpu             1
 22           DONE       RUN:cpu             1

Stats: Total Time 22
Stats: CPU Busy 12 (54.55%)
Stats: IO Busy  10 (45.45%)
```

5. [4] Now, run the same processes, but with the switching behavior set to switch to another process whenever one is WAITING for I/O (-l 2:0,8:100 -c -S SWITCH ON IO). What happens now? Use -c and -p to confirm that you are right.

When I run with SWITCH_ON_IO. Here, instead OS chooses process zero blocked state time to do other processes which in this case is process one.

```
(base) sneaky@sneaky-Lenovo-ideapad-520-15IKB:~/Documents/c/005_HA2/ostep-homework/cpu-intro$ ./process-run.py -l 2:0,8:100 -c -S SWITCH_ON_IO -c -p
Time        PID: 0        PID: 1           CPU           IOs
  1         RUN:io         READY             1
  2        BLOCKED       RUN:cpu             1             1
  3        BLOCKED       RUN:cpu             1             1
  4        BLOCKED       RUN:cpu             1             1
  5        BLOCKED       RUN:cpu             1             1
  6        BLOCKED       RUN:cpu             1             1
  7*         READY       RUN:cpu             1
  8          READY       RUN:cpu             1
  9          READY       RUN:cpu             1
 10    RUN:io_done          DONE             1
 11         RUN:io          DONE             1
 12        BLOCKED          DONE                           1
 13        BLOCKED          DONE                           1
 14        BLOCKED          DONE                           1
 15        BLOCKED          DONE                           1
 16        BLOCKED          DONE                           1
 17*   RUN:io_done          DONE             1

Stats: Total Time 17
Stats: CPU Busy 12 (70.59%)
Stats: IO Busy  10 (58.82%)
```

6. [4] 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 4:0,4:100,4:100,4:100 -S SWITCH ON IO -I IO RUN LATER -c -p) Are system resources being effectively utilized?

   With -S SWITCH ON IO -I IO RUN LATER, It takes 35 time units.
   Utilization is low which could be improved by making it IO RUN IMMEDIATE.

```
(base) sneaky@sneaky-Lenovo-ideapad-520-15IKB:~/Documents/c/005_HA2/ostep-homework/cpu-intro$ ./process-run.py -l 4:0,4:100,4:100,4:100 -S SWITCH_ON_IO -I IO_RUN_LATER -c
Time        PID: 0        PID: 1        PID: 2        PID: 3           CPU           IOs
  1         RUN:io         READY         READY         READY             1
  2        BLOCKED       RUN:cpu         READY         READY             1             1
  3        BLOCKED       RUN:cpu         READY         READY             1             1
  4        BLOCKED       RUN:cpu         READY         READY             1             1
  5        BLOCKED       RUN:cpu         READY         READY             1             1
  6        BLOCKED          DONE       RUN:cpu         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          DONE       RUN:cpu             1
 11          READY          DONE          DONE       RUN:cpu             1
 12          READY          DONE          DONE       RUN:cpu             1
 13          READY          DONE          DONE       RUN:cpu             1
 14    RUN:io_done          DONE          DONE          DONE             1
 15         RUN:io          DONE          DONE          DONE             1
 16        BLOCKED          DONE          DONE          DONE                           1
 17        BLOCKED          DONE          DONE          DONE                           1
 18        BLOCKED          DONE          DONE          DONE                           1
 19        BLOCKED          DONE          DONE          DONE                           1
 20        BLOCKED          DONE          DONE          DONE                           1
 21*   RUN:io_done          DONE          DONE          DONE             1
 22         RUN:io          DONE          DONE          DONE             1
 23        BLOCKED          DONE          DONE          DONE                           1
 24        BLOCKED          DONE          DONE          DONE                           1
 25        BLOCKED          DONE          DONE          DONE                           1
 26        BLOCKED          DONE          DONE          DONE                           1
 27        BLOCKED          DONE          DONE          DONE                           1
 28*   RUN:io_done          DONE          DONE          DONE             1
 29         RUN:io          DONE          DONE          DONE             1
 30        BLOCKED          DONE          DONE          DONE                           1
 31        BLOCKED          DONE          DONE          DONE                           1
 32        BLOCKED          DONE          DONE          DONE                           1
 33        BLOCKED          DONE          DONE          DONE                           1
 34        BLOCKED          DONE          DONE          DONE                           1
 35*   RUN:io_done          DONE          DONE          DONE             1

Stats: Total Time 35
Stats: CPU Busy 20 (57.14%)
Stats: IO Busy  20 (57.14%)
```

7. [4] 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?

Yes! It is better now as it is taking 28 time units. We know that initiating IO requests would take time to come in ready state. So it is a better idea to request in advance, and serve other processes meanwhile.

```
(base) sneaky@sneaky-Lenovo-ideapad-520-15IKB:~/Documents/c/OOS_HA2/ostep-homework/cpu-intro$ ./process-run.py -l 4:0,4:100,4:100,4:100 -S SWITCH_ON_IO -I IO_RUN_IMMEDIAT
Time        PID: 0        PID: 1        PID: 2        PID: 3           CPU           IOs
  1         RUN:io         READY         READY         READY             1
  2        BLOCKED       RUN:cpu         READY         READY             1             1
  3        BLOCKED       RUN:cpu         READY         READY             1             1
  4        BLOCKED       RUN:cpu         READY         READY             1             1
  5        BLOCKED       RUN:cpu         READY         READY             1             1
  6        BLOCKED          DONE       RUN:cpu         READY             1             1
  7*   RUN:io_done          DONE         READY         READY             1
  8         RUN:io          DONE         READY         READY             1
  9        BLOCKED          DONE       RUN:cpu         READY             1             1
 10        BLOCKED          DONE       RUN:cpu         READY             1             1
 11        BLOCKED          DONE       RUN:cpu         READY             1             1
 12        BLOCKED          DONE          DONE       RUN:cpu             1             1
 13        BLOCKED          DONE          DONE       RUN:cpu             1             1
 14*  RUN:io_done          DONE          DONE         READY             1
 15        RUN:io          DONE          DONE         READY             1
 16        BLOCKED          DONE          DONE       RUN:cpu             1             1
 17        BLOCKED          DONE          DONE       RUN:cpu             1             1
 18        BLOCKED          DONE          DONE          DONE                           1
 19        BLOCKED          DONE          DONE          DONE                           1
 20        BLOCKED          DONE          DONE          DONE                           1
 21*  RUN:io_done          DONE          DONE          DONE             1
 22        RUN:io          DONE          DONE          DONE             1
 23        BLOCKED          DONE          DONE          DONE                           1
 24        BLOCKED          DONE          DONE          DONE                           1
 25        BLOCKED          DONE          DONE          DONE                           1
 26        BLOCKED          DONE          DONE          DONE                           1
 27        BLOCKED          DONE          DONE          DONE                           1
 28*  RUN:io_done          DONE          DONE          DONE             1

Stats: Total Time 28
Stats: CPU Busy 20 (71.43%)
Stats: IO Busy  20 (71.43%)
```

8. [12] Now run with some randomly generated processes: -s 1 -l 4:50,4:50 or -s 2 -l 4:50,4:50 or -s 3 -l 4:50,4:50. 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?

   Now run with some randomly generated processes:

   -s 1 -l 4:50,4:50
   Stats: Total Time 18
   Stats: CPU Busy 12 (66.67%)
   Stats: IO Busy 15 (83.33%)

   -s 2 -l 4:50,4:50
   Stats: Total Time 25
   Stats: CPU Busy 13 (52.00%)
   Stats: IO Busy 18 (72.00%)

   -s 3 -l 4:50,4:50.
   Stats: Total Time 20
   Stats: CPU Busy 12 (60.00%)
   Stats: IO Busy 15 (75.00%)

-s 1:

SWITCH ON	IO RUN	TIME

SWITCH_ON_END	IO_RUN_LATER	32
SWITCH_ON_END	IO_RUN_IMMEDIATE	32
SWITCH_ON_IO	IO_RUN_LATER	18
SWITCH_ON_IO	IO_RUN_IMMEDIATE	18

-s 2:

SWITCH ON	IO RUN	TIME
SWITCH_ON_END	IO_RUN_LATER	38
SWITCH_ON_END	IO_RUN_IMMEDIATE	38
SWITCH_ON_IO	**IO_RUN_LATER**	**25**
SWITCH_ON_IO	IO_RUN_IMMEDIATE	26

-s 3:

SWITCH ON	IO RUN	TIME
SWITCH_ON_END	IO_RUN_LATER	32
SWITCH_ON_END	IO_RUN_IMMEDIATE	32
SWITCH_ON_IO	IO_RUN_LATER	20
SWITCH_ON_IO	IO_RUN_IMMEDIATE	20

Apparently, the best configuration is SWITCH ON IO, and RUN IO LATER. But from the results obtained before this question the best configuration is SWITCH ON IO, and RUN IO IMMEDIATE.

- SWITCH ON IO means whenever there is an IO request in the process switch to other processes like do not wait till this process request ends.
- RUN IO IMMEDIATE means that on switching prioritize to the process which initiates IO request, so those processes can come in ready states as soon as possible.