CSC373 Worksheet 1 Solution

August 16, 2020

1. The cpu utilization is 100%.

   The CPU utilization formula is given as

   $$\text{CPU Utilization} = 1 - \prod_i \text{I/O blocked time of ith process} \tag{1}$$

   Since the processes do no I/O, we can write there is no I/O blocked time.

   Thus, we can conclude

   $$\begin{aligned} \text{CPU Utilization} &= 1 - 0 \qquad (2) \\ &= 1 \qquad (3) \end{aligned}$$

   which is 100%.

   Notes

   - **CPU Utilization**
     - Means % of time CPU is in use
     - Formula is

   $$\text{CPU Utilization} = 1 - \prod_i \text{I/O blocked time of ith process} \tag{4}$$

   - **Process**
     - Means a program in execution

- **PID**
  - Is a short hand form for 'process identifier'
- **Process States**
  - in simplified view, process can be in one of the three states
    1. **Running:**
       * Is running on a processor
       * Means 'Is executing instructions'
    2. **Ready:**
       * Is ready to run
       * But, OS chosen to not to run it at the moment
    3. **Blocked:**
       * Is not ready to run until some other event takes place

       Example

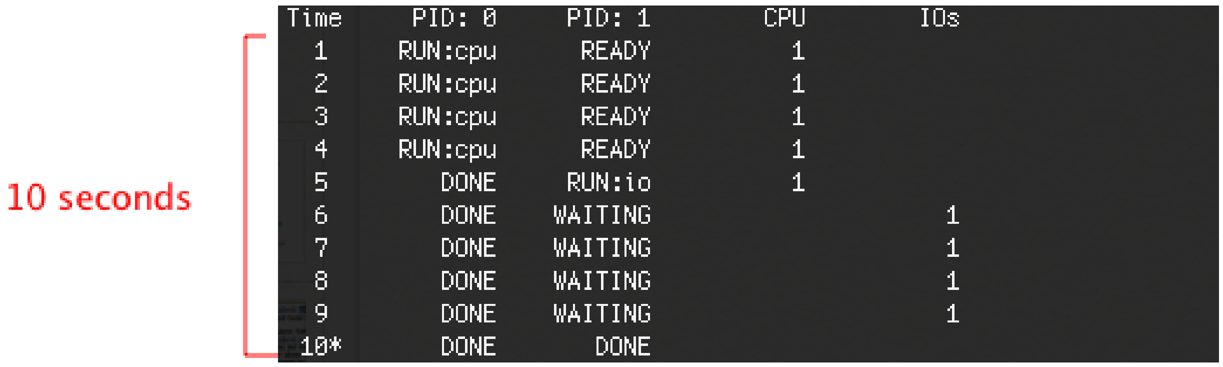
       Running an I/O request to disk → process blocked → other process can do their job while waiting

2. It takes total of 10 seconds to run.

   The first task only uses CPU, and takes 4 seconds.

   But, for the second task, on top of 4 seconds used for I/O, 1 second is used for preparing and initiating I/O, and the other 1 second is used for signaling that I/O is done.

   So in total, we have $4 + 4 + 1 + 1 = 10$ seconds.



```
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       DONE
```

3. Yes. Switching the order does matter.

   When the order is switched, the process 2 with I/O runs, and the process 2 enters the blocked state.

While at blocked state, the other process executes.

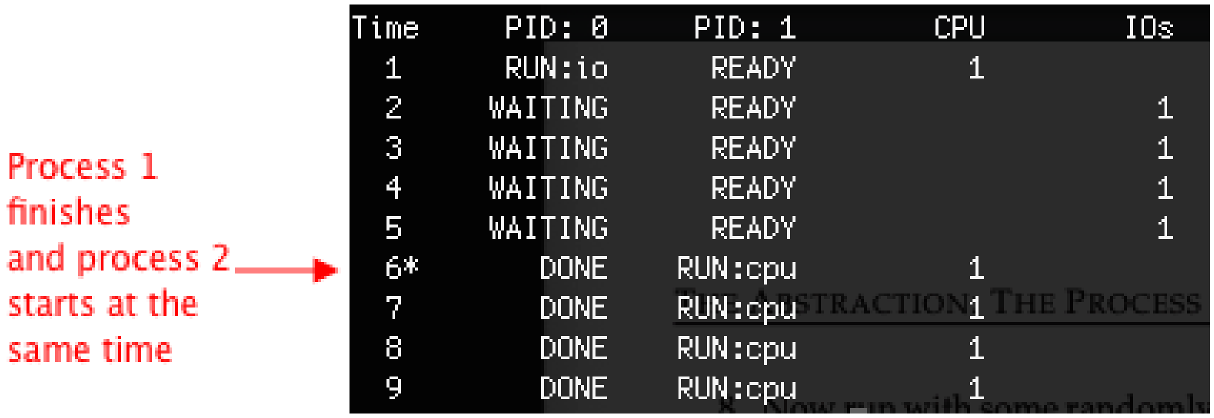
Since both take 4 seconds, by the time process 2 finishes, process 1 is finished.

Thus, total of 6 seconds are taken.

4. With flag SWITCH_ON_END, system runs as if it's without I/O. That is, process 2 runs after process 1 finishes.

   The only difference is that process 2 executes at the same time process 1 finishes.

   So instead of 10 seconds, there are 9 seconds in total



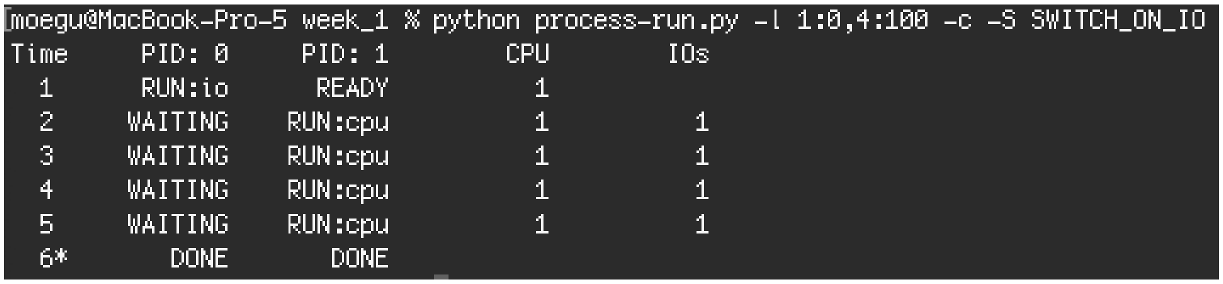
Process 1 finishes and process 2 starts at the same time

```
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*       DONE    RUN:cpu           1
  7        DONE    RUN:cpu           1
  8        DONE    RUN:cpu           1
  9        DONE    RUN:cpu           1
```

5. I need to write what happens when one is waiting for I/O (SWITCH_ON_IO).

   The result is the same as question 2.

   While process 1 is in blocked state, process 2 is executes.



```
moegu@MacBook-Pro-5 week_1 % python process-run.py -l 1:0,4:100 -c -S SWITCH_ON_IO
Time      PID: 0      PID: 1         CPU         IOs
  1       RUN:io       READY           1
  2      WAITING     RUN:cpu           1           1
  3      WAITING     RUN:cpu           1           1
  4      WAITING     RUN:cpu           1           1
  5      WAITING     RUN:cpu           1           1
  6*        DONE        DONE
```

6. First, I need to write what happens when combination of processes (-I IO_RUN_LATER, SWITCH_ON_IO) are used.

   There are total of four processes.

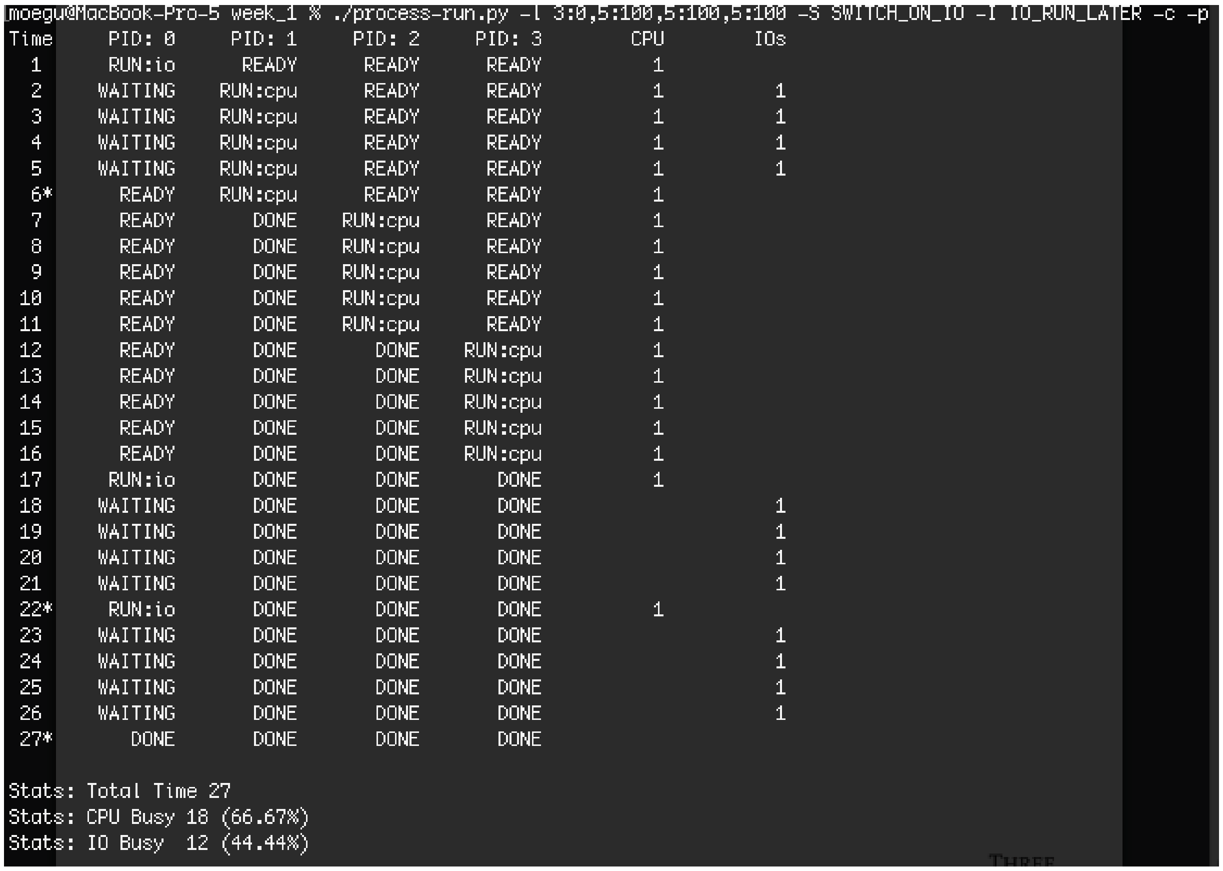
While process 1 is in blocked state for I/O, process 2 executes.

When process 1 finishes its first I/O operation, it doesn't execute the next right away. It waits for process 3 and 4 to finish until it finally gets its turn for more I/O operations.

Second, I need to write if the system resources are effectively utilized uder the combination of processes.

The answer is no.

System resources could have been utilized more effectively if process 3 and 4 are run while process 1 is performing it's I/O operation.



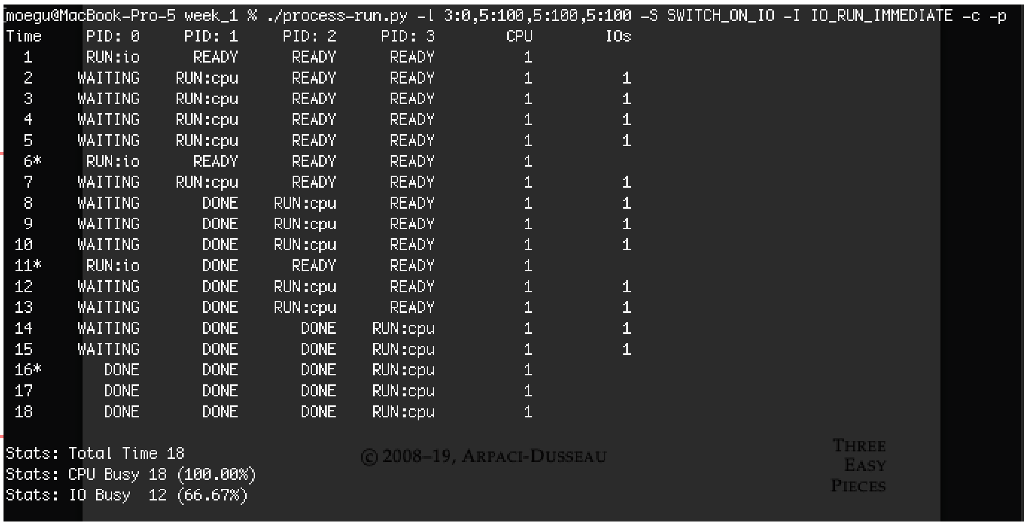
```
moegu@MacBook-Pro-5 week_1 % ./process-run.py -l 3:0,5:100,5:100,5:100 -S SWITCH_ON_IO -I IO_RUN_LATER -c -p
Time     PID: 0     PID: 1     PID: 2     PID: 3        CPU        IOs
  1      RUN:io      READY      READY      READY          1
  2     WAITING    RUN:cpu      READY      READY          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*      READY    RUN:cpu      READY      READY          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          1
 18     WAITING       DONE       DONE       DONE                     1
 19     WAITING       DONE       DONE       DONE                     1
 20     WAITING       DONE       DONE       DONE                     1
 21     WAITING       DONE       DONE       DONE                     1
 22*     RUN:io       DONE       DONE       DONE          1
 23     WAITING       DONE       DONE       DONE                     1
 24     WAITING       DONE       DONE       DONE                     1
 25     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%)
```

7. First, I need to write the difference between the process with `-I IO_RUN_LATER` and `-I IO_RUN_IMMEDIATE`.

   When the process is run with `-I IO_RUN_IMMEDIATE`, process 1 runs immediately one after another. And in each of process 1's blocked state, other processes are executed (process 2, process 3, process 4).

   This differs from `-I IO_RUN_LATER` where process 1 waits until other processes finish.

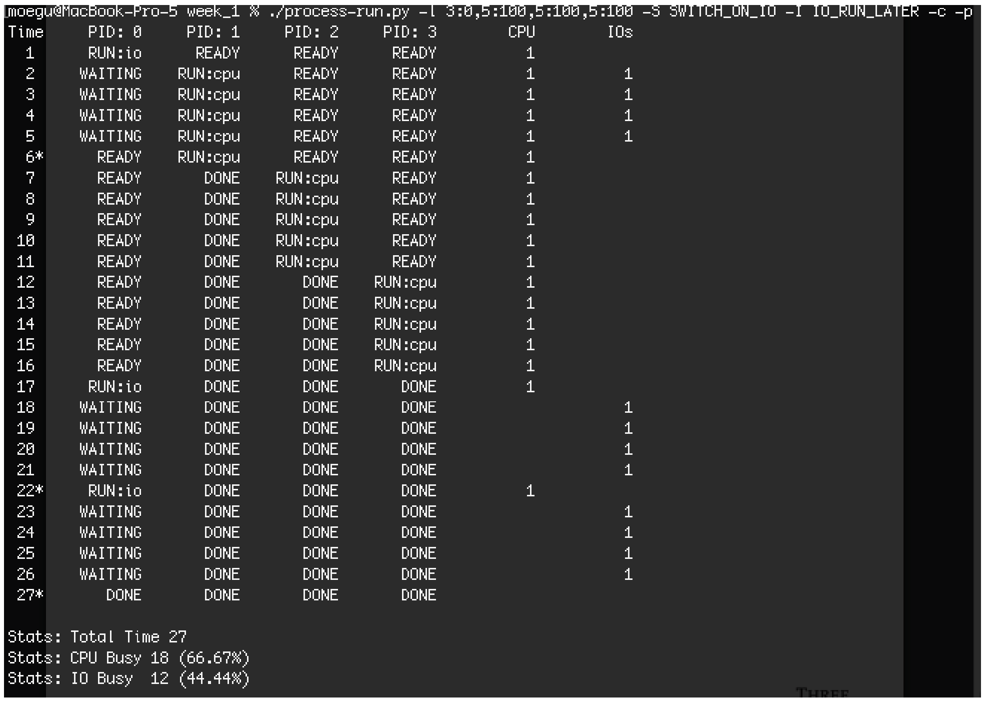
-I IO_RUN_IMMEDIATE



```
moegu@MacBook-Pro-5 week_1 % ./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*        RUN:io         READY         READY         READY             1
  7        WAITING       RUN:cpu         READY         READY             1             1
  8        WAITING          DONE       RUN:cpu         READY             1             1
  9        WAITING          DONE       RUN:cpu         READY             1             1
 10        WAITING          DONE       RUN:cpu         READY             1             1
 11*        RUN:io          DONE         READY         READY             1
 12        WAITING          DONE       RUN:cpu         READY             1             1
 13        WAITING          DONE       RUN:cpu         READY             1             1
 14        WAITING          DONE          DONE       RUN:cpu             1             1
 15        WAITING          DONE          DONE       RUN:cpu             1             1
 16*          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%)
```

Process 2,3,4 runs while process 1 is in blocked state

-I IO_RUN_LATER



```
moegu@MacBook-Pro-5 week_1 % ./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*         READY       RUN:cpu         READY         READY             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             1
 18        WAITING          DONE          DONE          DONE                           1
 19        WAITING          DONE          DONE          DONE                           1
 20        WAITING          DONE          DONE          DONE                           1
 21        WAITING          DONE          DONE          DONE                           1
 22*        RUN:io          DONE          DONE          DONE             1
 23        WAITING          DONE          DONE          DONE                           1
 24        WAITING          DONE          DONE          DONE                           1
 25        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%)
```

Process 1 waits until others finish

Second, I need to write why running a process that just completed an I/O again is a good idea?

It is a good idea since processes are better managed. That is, more can be done in less amount of time.

8. I need to write what happens when the following flags are used

- `-I IO_RUN_IMMEDIATE` vs `-I IO_RUN_LATER`

– `-s 1 -l 3:50,3:50`

When it is run with `-I IO_RUN_IMMEDIATE`, the CPU part of process 1 executes while process 2 waits in ready state.

When process 1 enters blocked state for I/O, process 2 starts.

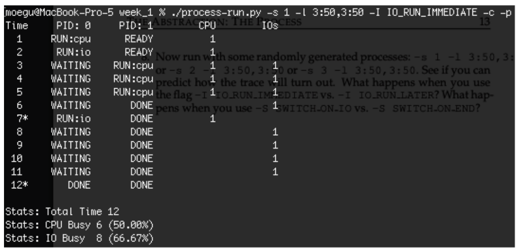
We know that process 1 stays in blocked state for 4 seconds for each I/O operation.

Since process 2 is all about CPU and takes 3 seconds to complete, process 2 will finish before process 1's second I/O operation.

Now, when it is run with `-I IO_RUN_LATER`, the same happens as above.

So, in this example, there is no difference between `-I IO_RUN_IMMEDIATE` and `-I IO_RUN_LATER`.

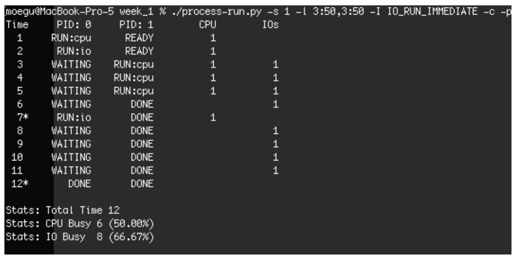
–I IO_RUN_LATER



```
moegu@MacBook-Pro-5 week_1 % ./process-run.py -s 1 -l 3:50,3:50 -I IO_RUN_IMMEDIATE -c -p
Time      PID: 0     PID: 1        CPU        IOs
  1      RUN:cpu      READY          1
  2       RUN:io      READY          1
  3      WAITING    RUN:cpu          1          1
  4      WAITING    RUN:cpu          1          1
  5      WAITING    RUN:cpu          1          1
  6      WAITING       DONE                     1
  7*      RUN:io       DONE          1
  8      WAITING       DONE                     1
  9      WAITING       DONE                     1
 10      WAITING       DONE                     1
 11      WAITING       DONE                     1
 12*        DONE       DONE

Stats: Total Time 12
Stats: CPU Busy 6 (50.00%)
Stats: IO Busy  8 (66.67%)
```

=

–I IO_RUN_IMMEDIATE



```
moegu@MacBook-Pro-5 week_1 % ./process-run.py -s 1 -l 3:50,3:50 -I IO_RUN_IMMEDIATE -c -p
Time      PID: 0     PID: 1        CPU        IOs
  1      RUN:cpu      READY          1
  2       RUN:io      READY          1
  3      WAITING    RUN:cpu          1          1
  4      WAITING    RUN:cpu          1          1
  5      WAITING    RUN:cpu          1          1
  6      WAITING       DONE                     1
  7*      RUN:io       DONE          1
  8      WAITING       DONE                     1
  9      WAITING       DONE                     1
 10      WAITING       DONE                     1
 11      WAITING       DONE                     1
 12*        DONE       DONE

Stats: Total Time 12
Stats: CPU Busy 6 (50.00%)
Stats: IO Busy  8 (66.67%)
```

– `-s 2 -l 3:50,3:50`

When it is run with `-I IO_RUN_IMMEDIATE`, process 1 enters blocked state for I/O operation at time = 2, and process 2 executes while process 1 is in blocked state between time = 2 and time = 5.

We know that each CPU operation takes 1 second and initialization of I/O operation takes 1 second.

Using this information, process 2 will enter blocked state for I/O operation at time = 3 until time = 6.

Then, at time = 6, process 1 will run another I/O operation and enter blocked state from time = 7 to time = 10.

Then, at time = 8, process 2 will run last I/O operation and enter blocked state from time = 9 to time = 12.

Then, at time = 11, process 1 will execute CPU operation, and will finish at the same time.

Now, when it is run with `-I IO_RUN_LATER`, the same happens as above.

So, in this example, there is no difference between `-I IO_RUN_IMMEDIATE` and `-I IO_RUN_LATER`.

- `-S SWITCH_ON_IO` vs `-S SWITCH_ON_END`