Process Simulation Homework Solutions with Explanations

# Question 1

Run process-run.py with the following flags: -l 5:100,5:100.

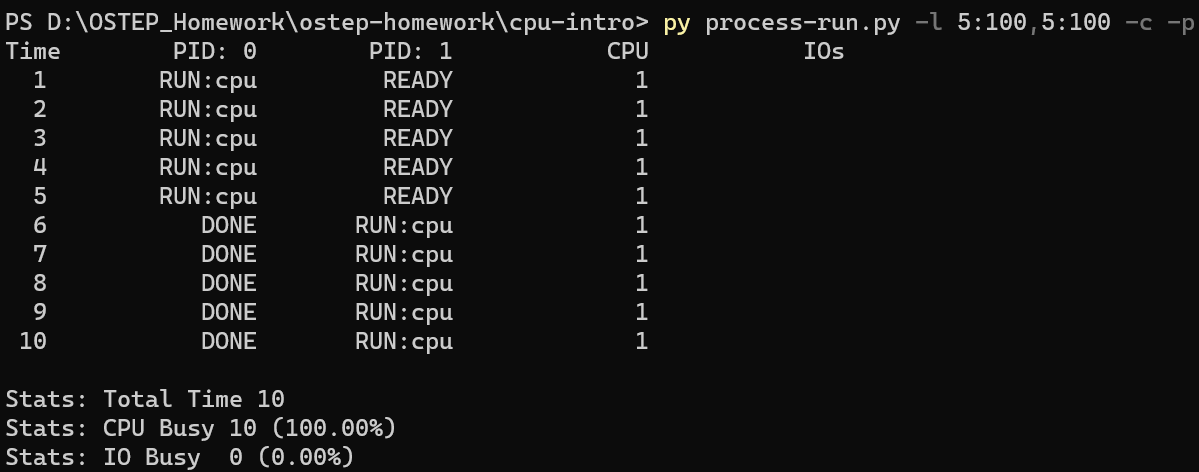
## Answer:

CPU utilization is 100%.

## Explanation:

Both processes only perform CPU instructions. With no I/O requests, the CPU never idles. Therefore, CPU utilization remains 100%.

## Output:



# Question 2

Run with ./process-run.py -l 4:100,1:0.

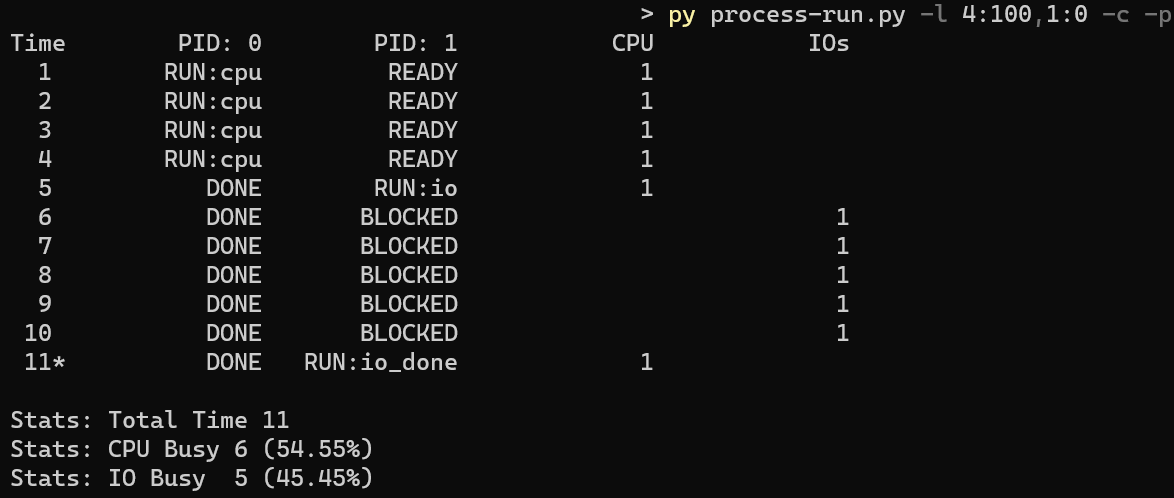
## Answer:

Total time is 11 units.

## Explanation:

Process A completes its 4 CPU instructions while Process B immediately issues I/O and waits. After A finishes, the system waits for the I/O to complete (about 5 ticks). Thus total completion time is ~11.

## Output:



# Question 3

Switch the order of processes: -l 1:0,4:100.

## Answer:

Total time is 7 units.

## Explanation:

Process A issues I/O first and blocks. While I/O runs, Process B executes 4 CPU instructions. By the time B finishes, A’s I/O has also completed. CPU and I/O overlap reduces total runtime to about 7.

## Output:

# Question 4

Run with -l 1:0,4:100 -c -S SWITCHONEND.

## Answer:

CPU utilization drops due to idle waiting.

## Explanation:

With SWITCHONEND, the CPU does not switch to another process when one issues I/O. It idles until the I/O completes, then runs the CPU process. This wastes cycles and reduces utilization.

## Output:

# Question 5

Run with -l 1:0,4:100 -c -S SWITCHONIO.

## Answer:

CPU and I/O overlap efficiently.

## Explanation:

SWITCHONIO lets the CPU switch to another process immediately when one is waiting for I/O. Thus, the CPU process runs during I/O wait, and utilization is maximized.

## Output:

# Question 6

Run with ./process-run.py -l 3:0,5:100,5:100,5:100 -S SWITCHONIO -c -p -I IORUNLATER.

## Answer:

System is efficiently utilized, but I/O process response time increases.

## Explanation:

With IORUNLATER, when I/O completes, the process is not run immediately but waits in the ready queue. CPU jobs continue first. Utilization remains high, but I/O-bound processes face delays.

## Output:

# 

# 

# Question 7

Run with ./process-run.py -l 3:0,5:100,5:100,5:100 -S SWITCHONIO -c -p -I IORUNIMMEDIATE.

## Answer:

I/O process resumes immediately after I/O completes.

## Explanation:

With IORUNIMMEDIATE, as soon as an I/O finishes, the process that issued it preempts and resumes execution. This improves responsiveness of I/O-bound processes, which is good for interactive tasks.

## Output:

# Question 8

Run with randomly generated processes: -s 1 -l 3:50,3:50 (and variations).

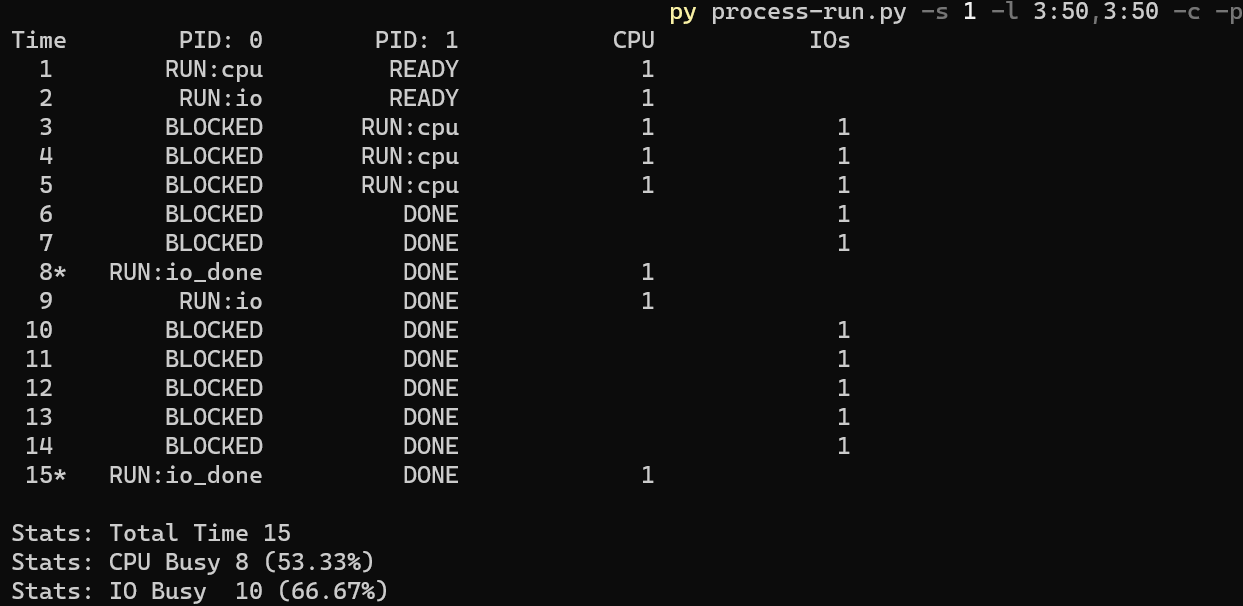
## Answer:

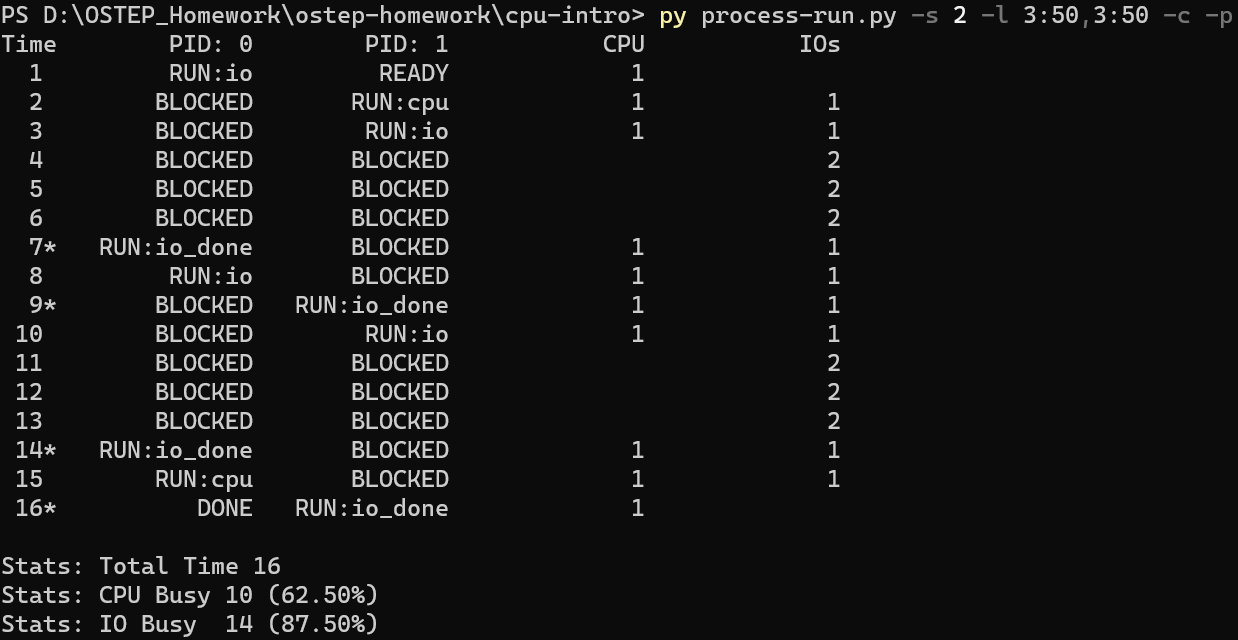
Behavior depends on scheduling flags.

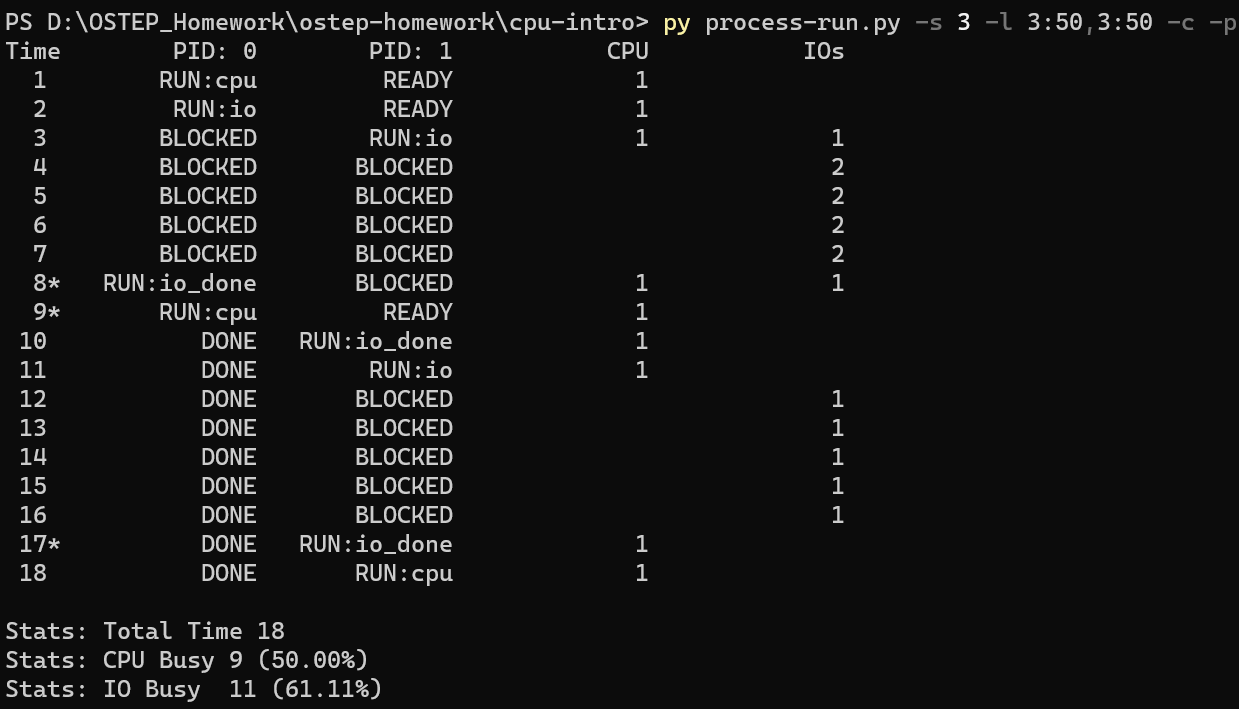
## Explanation:

With 50% CPU and 50% I/O instructions, outcomes vary. IORUNIMMEDIATE gives faster I/O response, IORUNLATER delays I/O jobs. SWITCHONIO maximizes CPU usage during waits, SWITCHONEND may leave CPU idle. These differences are visible in the traces.

## Output:







## Competitive Analysis:

| **Scenario** | **Behavior** | **Effect on CPU Utilization** | **Effect on Turnaround Time** | **Explanation** |
| --- | --- | --- | --- | --- |
| **-I IORUNLATER** | After I/O finishes, process waits until CPU is free. | CPU keeps working on whatever was running, so utilization is **good**, but I/O-heavy processes may be **delayed**. | Longer for I/O-heavy jobs. | Efficient for throughput, not great for responsiveness. |
| **-I IORUNIMMEDIATE** | After I/O finishes, that process **immediately resumes**. | CPU sometimes switches rapidly (context switches), but I/O-heavy processes finish **sooner**. | Better response time for I/O-bound processes. | Useful when interactive jobs are important. |
| **-S SWITCH\_ON\_END** | If a process does I/O, the CPU just **waits**, no switch. | CPU sits idle → **low utilization**. | Bad for overall system performance. | Simple but wasteful. |
| **-S SWITCH\_ON\_IO** | When process does I/O, scheduler runs another process. | CPU is almost always busy → **high utilization**. | Both CPU and I/O tasks finish faster. | Best use of resources. |