# CSC373 Worksheet 1 Solution

# August 16, 2020

## 1. The cpu utilization is 100%.

The CPU utilization formula is given as

CPU Utilization = 
$$1 - \prod_{i} I/O$$
 blocked time of ith process (1)

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

Thus, we can conclude

$$CPU Utilization = 1 - 0$$

$$= 1$$
(2)

which is 100%.

## Notes

# • CPU Utilization

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

CPU Utilization = 
$$1 - \prod_{i} I/O$$
 blocked time of ith process (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  $\rightarrow$  process blocked  $\rightarrow$  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	IUs	
	1	RUN:cpu	READY	1		
	2	RUN:cpu	READY	1		
	3	RUN:cpu	READY	1		
	4	RUN:cpu	READY	1		
0 seconds	5	DONE	RUN:io	1		
o seconus	6	DONE	WAITING		1	
	7	DONE	WAITING		1	
	8	DONE	WAITING		1	
	<b>9</b>	DONE	WAITING		1	
	10*	DONE	DONE			

## 10

## 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

	Time	PID: 0	PID: 1	CPU	IOs
	1	RUN:io	READY	1	
	2	WAITING	READY		1
Process 1	3	WAITING	READY		1
	4	WAITING	READY		1
finishes	5	WAITING	READY		1
and process 2	6*	DONE	RUN:cpu	1	
starts at the	7	DONE	RUN:cpustr	ACTION1 THE	PROCESS
same time	8	DONE	RUN:cpu	1	
	9	DONE	RUN:cpu	run with some	randomly

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@M	1acBook–Pro-	-5 week_1 %	ያ python proces:	s–run.py -	-l 1:0,4:100	-c -S	SWITCH_ON_IO
	Time	PID: 0	PID: 1	CPU	IOs			
	1	RUN:io	READY	1				
1	2	WAITING	RUN:cpu	1	1			
1	3	WAITING	RUN:cpu	1	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.

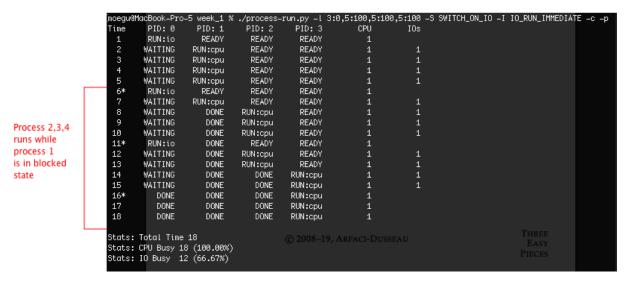
moegu	u@MacBook−Pro	-5 week_1 %	% ./process	-run.py –l	3:0,5:100,5	:100,5:100	_S SWITCH_ON_IO _I	IO_RUN_LATER -c -p	
Time	PID: 0	PID: 1	PID: 2	PID: 3	CPU	I0s			
1	RUN:io	READY	READY	READY	1				
2	WAITING	RUN:cpu	READY	READY	1	1			
3	WAITING	RUN:cpu	READY	READY	1	1			
4	WAITING	RUN:cpu	READY	READY	1	1			
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%)									
State	s: IO Busy 1	.2 (44.44%)					-		
								HEREE	

7. First, I need to write the difference between the process with -I IO\_RUN\_LATER and -I IO\_RUN\_IMMEDIATE.

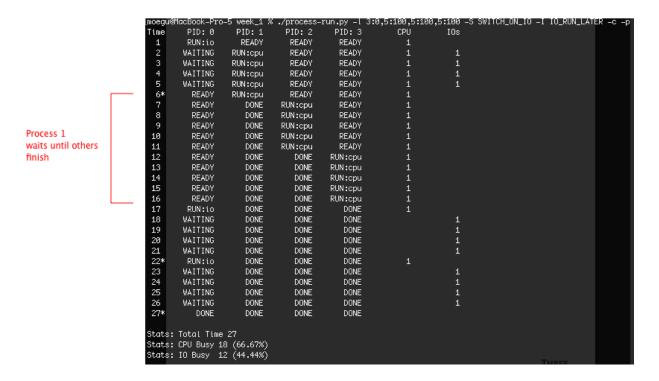
When the process is run with -I IO\_RUN\_IMMEDIATE, process 1 runs immediately one after another. And in each of process 1's blocked state, other processes are executed (process 2, process 3, process 4).

This differs from -I IO\_RUN\_LATER where process 1 waits until other processes finish.

#### -I IO\_RUN\_IMMEDIATE



#### -I IO\_RUN\_LATER



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 -1 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.

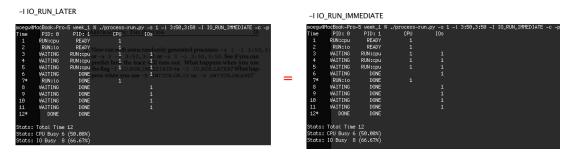
When process 1 enters blocked state for I/O, process 2 starts.

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.



--s 2 -1 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

$$--s$$
 3  $-1$  3:50,3:50

When the processes are run with -S SWITCH\_ON\_IO, it will work the same as when ran with -I IO\_RUN\_IMMEDIATE.

However, when the processes are run with -S SWITCH\_ON\_END, process 2 will begin when process 1 finishes.

And this happens at time = 8.

