

Dokumentation:

1. What should the CPU utilisation be? Why do you know this?

→ It should be 100%. I know this because there is 100% chance that an instruction will be done on CPU

```
vladb@VladB ~/G/h/S/B/H/HW1_processes (master)> ./process-run.py -l 5:100,5:100
```

Produce a trace of what would happen when you run these processes:

Process 0

cpu
cpu
cpu
cpu
cpu

Process 1

cpu
cpu
cpu
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)

```
vladb@VladB ~/G/h/S/B/H/HW1_processes (master)>
```

```
./process-run.py -l 5:100,5:100 -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         DONE    RUN:cpu        1
7         DONE    RUN:cpu        1
8         DONE    RUN:cpu        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. How long does it take to complete both processes?

→ It should take 10 clock ticks, 4 for the process that runs only on CPU, 5 (default IO length) for the process that runs on IO (first one is run on CPU), and 1 for executing the instruction after finishing i/o

```
vladb@VladB ~/G/h/S/B/H/HW1_processes (master)> ./process-run.py -l 4:100,1:0
```

Produce a trace of what would happen when you run these processes:

Process 0

```

cpu
cpu
cpu
cpu

```

Process 1

```
io
```

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)

```
vladb@VladB ~/G/h/S/B/H/HW1_processes (master)>
```

```
./process-run.py -l 4:100,1:0 -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	DONE	RUN:io	1	
6	DONE	WAITING		1
7	DONE	WAITING		1
8	DONE	WAITING		1
9	DONE	WAITING		1
10*	DONE	DONE		

Stats: Total Time 10

Stats: CPU Busy 5 (50.00%)

Stats: IO Busy 4 (40.00%)

3. Does switching the order matter? Why?

→ The order does matter. The IO of the first process can happen during the execution of the second process which runs only on CPU. So now the first instruction starts the IO, then 4 clock ticks for the second process, then 1 for executing the instruction of first process.

→ 6 clock ticks

```
vladb@VladB ~/G/h/S/B/H/HW1_processes (master) [2]>
```

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

Produce a trace of what would happen when you run these processes:

Process 0

io

Process 1

cpu

cpu

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)

```
vladb@VladB ~/G/h/S/B/H/HW1_processes (master)>
```

```
./process-run.py -l 1:0,4:100 -c -p
```

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		

```
Stats: Total Time 6
```

```
Stats: CPU Busy 5 (83.33%)
```

```
Stats: IO Busy 4 (66.67%)
```

Warum ist 7. Schritt PID:0 RUNNING und nicht DONE?

```
vladb@VladB ~/G/h/S/B/H/HW1_processes (master)> ./process-run.py -l 1:0,5:100 -c
```

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*	READY	RUN:cpu	1	
7	RUNNING	DONE		

Weil default von -l ist IO_RUN_LATER

```
vladb@VladB ~/G/h/S/B/H/HW1_processes (master)>
```

```
./process-run.py -l 1:0,5:100 -I IO_RUN_IMMEDIATE -c
```

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	RUN: cpu	1	

4. 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?

→ Would have expected 10 clock ticks, but it is actually 9. 1 for the start of IO, 4 for IO, and then 4 for the process on CPU. I think that it is 9 now, because in 2. FRAGE the I/O was at the end and it needed on clock tick more to change the state of the process to DONE. In this example, the change of state from WAITING to DONE can happen on the execution of the first instruction of the second process. This last clock tick does NOT run on CPU. See graph

```
vladb@VladB ~/G/h/S/B/H/HW1_processes (master)>
./process-run.py -l 1:0,4:100 -S SWITCH_ON_END -c
```

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. (-l 1:0,4:100 -c -S SWITCH_ON_IO) What happens now?

- It needs 6 clock ticks
- 1 for starting the I/O
- 4 for the second process on CPU. In the background 4 for I/O
- 1 for changing the state of first process to DONE
- Same result as ./process-run.py -l 1:0,4:100 -c -p

→ Default ist SWITCH_ON_IO

```
vladb@VladB ~/G/h/S/B/H/HW1_processes (master)>
./process-run.py -l 1:0,4:100 -S SWITCH_ON_IO -c
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. What happens when you run this combination of processes? (Run `./process-run.py -l 3:0,5:100,5:100,5:100 -S SWITCH ON IO -I IO RUN LATER -c -p`)

Are system resources being effectively utilised?

→ The first *IO instruction of process 1 is called, and the IO* happens during the execution of process 2. Then process 3 and 4 follow. At the end the rest of 2 instructions of process 1 happen. In this time the CPU is not used. SO the resources are not being effectively used.

→ Default ist IO_RUN_LATER

```

vladb@VladB ~/G/h/S/B/H/HW1_processes (master)>
./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      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      1
7      READY    DONE     RUN:cpu   READY    1      1
8      READY    DONE     RUN:cpu   READY    1      1
9      READY    DONE     RUN:cpu   READY    1      1
10     READY    DONE     RUN:cpu   READY    1      1
11     READY    DONE     RUN:cpu   READY    1      1
12     READY    DONE     DONE     RUN:cpu   1      1
13     READY    DONE     DONE     RUN:cpu   1      1
14     READY    DONE     DONE     RUN:cpu   1      1
15     READY    DONE     DONE     RUN:cpu   1      1
16     READY    DONE     DONE     RUN:cpu   1      1
17     RUN:io    DONE     DONE     DONE     1      1
18     WAITING  DONE     DONE     DONE     1      1
19     WAITING  DONE     DONE     DONE     1      1
20     WAITING  DONE     DONE     DONE     1      1
21     WAITING  DONE     DONE     DONE     1      1
22*    RUN:io    DONE     DONE     DONE     1      1
23     WAITING  DONE     DONE     DONE     1      1
24     WAITING  DONE     DONE     DONE     1      1
25     WAITING  DONE     DONE     DONE     1      1
26     WAITING  DONE     DONE     DONE     1      1
27*    DONE     DONE     DONE     DONE     1      1

Stats: Total Time 27
Stats: CPU Busy 18 (66.67%)
Stats: IO Busy 12 (44.44%)

```

7. How does this behaviour differ? Why might running a process that just completed an I/O again be a good idea?

→ Now there are only 18 clock ticks needed, instead of 27. And the CPU is used 100% of the time! What happens is, that after each of the processes 2, 3 the IO instructions of the first process are called. During the time the CPU executes processes 1, 2 and 3, the IO is also done. It is a good idea, because there might be another I/O instruction, and its initialisation could be done during execution of other processes.

```

vladb@VladB ~/G/h/S/B/H/HW1_processes (master)>
./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      0
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      0
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      0
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      0
17     DONE     DONE     DONE     RUN:cpu   1      0
18     DONE     DONE     DONE     RUN:cpu   1      0

Stats: Total Time 18
Stats: CPU Busy 18 (100.00%)
Stats: IO Busy 12 (66.67%)

```

8. 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?

→ IO RUN IMMEDIATE should be better, because when it comes back from IO, it should run other IO as soon as possible.

→ SWITCH ON IO is better, because you can do IO tasks in the background

-I IO RUN IMMEDIATE vs. -I IO RUN LATER

SEED 1

```

vladb@VladB ~/G/h/S/B/H/HW1_processes (master)>
./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      0
2      RUN:io    READY    1      0
3      WAITING  RUN:cpu   1      1
4      WAITING  RUN:cpu   1      1
5      WAITING  RUN:cpu   1      1
6      WAITING  DONE     1      1
7*     RUN:io    DONE     1      0
8      WAITING  DONE     1      1
9      WAITING  DONE     1      1
10     WAITING  DONE     1      1
11     WAITING  DONE     1      1

```


12* DONE DONE

Stats: Total Time 12

Stats: CPU Busy 6 (50.00%)

Stats: IO Busy 8 (66.67%)

vladb@VladB ~/G/h/S/B/H/HW1_processes (master)>

./process-run.py -s 1 -l 3:50,3:50 -I IO_RUN_LATER -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%)