Tobias Sautter BSYS HW7 **20.03.2023**

Question 1

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

CPU utilization should be 100% since you don't have any I/O you need to wait for.

```
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%)
```

Question 2

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

```
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       BLOCKED                           1
  7           DONE       BLOCKED                           1
  8           DONE       BLOCKED                           1
  9           DONE       BLOCKED                           1
 10           DONE       BLOCKED                           1
 11*          DONE   RUN:io_done             1

Stats: Total Time 11
Stats: CPU Busy 6 (54.55%)
Stats: IO Busy  5 (45.45%)
```

Question 3

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

It only takes 7 ticks since the 2nd process can run while the first is waiting for the I/O

```
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          DONE                           1
  7*   RUN:io_done          DONE             1

Stats: Total Time 7
Stats: CPU Busy 6 (85.71%)
Stats: IO Busy  5 (71.43%)
```

Question 4

We'll now explore some of the other flags. One important flag is -S, which determines how the system reacts when a process is- sues 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 1:0,4:100 -c -S SWITCH_ON_END), one doing I/O and the other doing CPU work?

Now it waits until the I/O is done before starting the next Process thus taking 11 ticks again.

```
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           DONE       RUN:cpu             1
  9           DONE       RUN:cpu             1
 10           DONE       RUN:cpu             1
 11           DONE       RUN:cpu             1

Stats: Total Time 11
Stats: CPU Busy 6 (54.55%)
Stats: IO Busy  5 (45.45%)
```

Question 5

Now, run the same processes, but with the switching behavior set to switch to another process whenever one is WAITING for I/O (-l 1:0,4:100 -c -S SWITCH_ON_IO). What happens now? Use -c and -p to confirm that you are right.

With the SWITCH_ON_IO option it is back to the 'natural' order as described in Q3

```
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          DONE                           1
  7*   RUN:io_done          DONE             1

Stats: Total Time 7
Stats: CPU Busy 6 (85.71%)
Stats: IO Busy  5 (71.43%)
```

Question 6

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 3:0,5:100,5:100,5:100 -S SWITCH_ON_IO -I IO_RUN_LATER -c -p) Are system resources being effectively utilized?

In this case it is not efficient. Since we flagged IO_RUN_LATER the IO is waiting for all other Processes to be finished rather then using the I/O Blocked time to run the other Processes and saving a lot of time.

```
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       RUN:cpu         READY         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       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          DONE             1
 18         RUN:io          DONE          DONE          DONE             1
 19        BLOCKED          DONE          DONE          DONE                           1
 20        BLOCKED          DONE          DONE          DONE                           1
 21        BLOCKED          DONE          DONE          DONE                           1
 22        BLOCKED          DONE          DONE          DONE                           1
 23        BLOCKED          DONE          DONE          DONE                           1
 24*   RUN:io_done          DONE          DONE          DONE             1
 25         RUN:io          DONE          DONE          DONE             1
 26        BLOCKED          DONE          DONE          DONE                           1
 27        BLOCKED          DONE          DONE          DONE                           1
 28        BLOCKED          DONE          DONE          DONE                           1
 29        BLOCKED          DONE          DONE          DONE                           1
 30        BLOCKED          DONE          DONE          DONE                           1
 31*   RUN:io_done          DONE          DONE          DONE             1

Stats: Total Time 31
Stats: CPU Busy 21 (67.74%)
Stats: IO Busy  15 (48.39%)
```

Question 7

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?

This is a way better usage of the CPU and the I/O, because every time the processor is waiting for the I/O to finish another process can be run in the meantime.
A process that just had an I/O is likely not going to use one immediately again

```
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       RUN:cpu         READY         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       RUN:cpu         READY             1             1
 13        BLOCKED          DONE       RUN:cpu         READY             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       RUN:cpu             1             1
 19        BLOCKED          DONE          DONE       RUN:cpu             1             1
 20        BLOCKED          DONE          DONE       RUN:cpu             1             1
 21*   RUN:io_done          DONE          DONE          DONE             1

Stats: Total Time 21
Stats: CPU Busy 21 (100.00%)
Stats: IO Busy  15 (71.43%)
```