**Homework 1, process-run.py**

**Due** Saturday by 11:59pm

**Points** 48

**Submitting** a text entry box

**Instructions**  
Please run process-run.py with the commands listed below. Use the terminal for this exercise in Github Codespaces. You will need to change the execute permissions for process-run.py. For each **numbered** question, provide:

* A **numeric** or **yes/no** answer (brief).
* A **1–2 sentence justification** referring to the trace details (-c or -p). Submit your answers directly in Canvas using the **text entry** field, and clearly label each numbered response.

1. CPU Utilization (6 points)

**Command**: ./process-run.py -l 5:100,5:100

A computer screen shot of a black screen

Description automatically generated

A screen shot of a computer

Description automatically generated

* (a) **Numeric (2 pts)**: What is the CPU utilization (in %)?

CPU Utilization = 100.00%

This is evident from the output: Stats: CPU Busy 10 (100.00%)

* (b) **Justification (4 pts)**: Why does this number make sense?

The trace shows that both processes (PID 0 and PID 1) only execute CPU instructions (RUN:cpu) without any I/O operations.

Since the CPU is continuously running throughout the **Total Time of 10** units, the utilization remains at **100%**.

The **IO Busy** time is **0 (0.00%)**, meaning no time was spent waiting on I/O.

2. Completion Time (6 points)

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

A screenshot of a computer

Description automatically generated

* (a) **Numeric (2 pts)**: How many time units does it take to finish both processes?

**Total completion time: 5 time units**

Process 0 executes **4 CPU instructions** (each taking 1 time unit).

Process 1 issues an **I/O operation**, which takes **1 additional time unit** (assuming default I/O duration).

* (b) **Justification (4 pts)**: Provide one specific trace detail that confirms your answer.

From the trace, **Process 0** executes 4 cpu instructions.

**Process 1** issues an io instruction, requiring additional time for completion.

The system switches when the current process **FINISHES** or **ISSUES AN IO**, meaning Process 1 executes **after** Process 0 completes.

This results in a total time of **5 units**.

3. Process Order (8 points)

Compare the outputs from:

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

A screenshot of a computer program

Description automatically generated

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

A screen shot of a computer

Description automatically generated

* (a) **Yes/No (3 pts)**: Does reversing the order change total completion time?

**Yes** (Total time changed from **10** to **6**).

* (b) **Justification (5 pts)**: Cite at least one reason from the trace explaining why (or why not).

In the **first case (4:100,1:0)**, Process 0 (CPU-heavy) runs first, completing all CPU instructions before Process 1 starts. Since Process 1 performs I/O, it must wait, leading to extra idle time.

In the **second case (1:0,4:100)**, Process 1 starts with an **I/O operation immediately**. While it waits, Process 0 efficiently runs its CPU instructions in parallel. This **reduces overall execution time from 10 to 6** because the CPU is utilized more efficiently, minimizing idle time.

4. Switching Policies (16 points total)

(a) SWITCH ON END (8 points)

**Command**: ./process-run.py -l 1:0,4:100 -S SWITCH\_ON\_END -c

A screen shot of a computer

Description automatically generated

* (a1) **Yes/No (2 pts)**: Does the scheduler switch away from a process performing I/O before it completes?

NO

* (a2) **Justification (6 pts)**: In 1–2 sentences, explain how SWITCH\_ON\_END enforces that behavior.

The scheduler follows the **SWITCH\_ON\_END** policy, meaning it does **not** switch processes when a process issues an I/O request. Instead, the process **must wait for its I/O to complete before the CPU switches to another process**.

In the trace, **Process 0 remains in WAITING state (Time Steps 2-5)** while its I/O completes, only after that does Process 1 begin execution at **Time Step 6**.

(b) SWITCH ON IO (8 points)

**Command**: ./process-run.py -l 1:0,4:100 -S SWITCH\_ON\_IO -c

A screen shot of a computer

Description automatically generated

* (b1) **Yes/No (2 pts)**: Does the scheduler switch processes as soon as one issues an I/O request?

Yes

* (b2) **Justification (6 pts)**: Contrast SWITCH\_ON\_IO with SWITCH\_ON\_END in 1–2 sentences.

The **SWITCH\_ON\_IO** policy **immediately switches to another process** when a process initiates an I/O operation.

In the trace, after **Process 0 starts I/O at Time Step 1**, the scheduler **immediately switches to Process 1** at **Time Step 2**.

This approach is **more efficient** because it allows the CPU to keep working while the I/O operation is in progress, reducing overall idle time.

5. I/O Completion Policies (12 points total)

(a) IO\_RUN\_LATER (6 points)

**Command**:

./process-run.py -l 3:0,5:100,5:100,5:100 \ -S SWITCH\_ON\_IO -I IO\_RUN\_LATER -c -p

A screenshot of a computer

Description automatically generated

* (a1) **Yes/No (2 pts)**: Does the CPU become idle while a process that completed I/O remains unscheduled?

Yes

* (a2) **Justification (4 pts)**: How does this affect CPU utilization?

The **IO\_RUN\_LATER** policy means that when a process completes its I/O operation, it does **not** immediately get scheduled to run. Instead, it waits until the scheduler chooses it again in its turn.

In the trace:

At **Time 1**, PID 0 starts an **I/O operation**.

At **Time 2-5**, PID 0 is in **WAITING state**, while the CPU runs another process.

Later at **Time 20-21**, another I/O operation is issued, but **the CPU remains idle while waiting for it to be scheduled again**.

This behavior leads to periods where no CPU execution happens, **reducing CPU utilization**.

(b) IO\_RUN\_IMMEDIATE (6 points)

**Command**:

./process-run.py -l 3:0,5:100,5:100,5:100 \ -S SWITCH\_ON\_IO -I IO\_RUN\_IMMEDIATE -c -p

A screenshot of a computer

Description automatically generated

* (b1) **Yes/No (2 pts)**: Does the just-finished I/O process run immediately?

Yes

* (b2) **Justification (4 pts)**: Why might this improve overall performance?

The **IO\_RUN\_IMMEDIATE** policy ensures that when a process finishes its I/O operation, it is immediately scheduled to run without waiting for the scheduler’s next cycle.

In the trace:

At **Time 1**, PID 0 starts an **I/O operation**.

At **Time 6**, PID 0 completes I/O and immediately resumes execution.

The same behavior is observed at **Time 11** and **Time 16**, where as soon as an I/O operation finishes, the process runs immediately.

This behavior **improves CPU utilization** because there are **no idle cycles waiting for the scheduler to pick the process again**.

Comparing with **IO\_RUN\_LATER**, the **total time is reduced from 27 (IO\_RUN\_LATER) to 18 (IO\_RUN\_IMMEDIATE)**, showing that this policy significantly improves efficiency.