Assignment 2: Scheduling Policies

Professor: Dalu Jacob
TA: Anshika Prajapati, Priyal Jain, Saket Kandoi

Deadline: 15th October 11:59 p.m

Part 1: Offline Scheduling

See `offline_schedulers.h` for reference. Time slices and boost times are specified as variables in the header file.
Input: List of processes which will be provided before the program starts.
Implement the following CPU scheduling algorithms:

1. First-Come First-Serve (FCFS)
2. Round Robin (RR). Assume order of processes provided will be the order required in the first iteration.
3. Multi-level Feedback Queue (MLFQ) with 3 queues. Assume order of processes provided will be the order required in the first iteration, and all processes are initially added to highest priority queue.

Input for FCFS:

```
Process {cmd1, cmd2, cmd3}
```

Expected Behaviour:

```
Executes cmd1, context switch when it terminates print the results.
Executes cmd2, context switch when it terminates print the results.
Executes cmd3, context switch when it terminates print the results.
Then terminate the scheduler on completion of all processes.
```

Part 2: Online Scheduling

See `online_schedulers.h` for reference. Time slices and boost times are specified as variables in the header file.
Input: Processes will be entered dynamically in real- time while the program runs. After each context switch, the scheduler must check the terminal for any new commands. Every line ending with a newline character (`'\n'`) should be treated as the arrival of a new process.
The following CPU scheduling techniques are to be implemented:

1. Multi-level Feedback Queue (MLFQ) with adaptive behavior

(a) **Initial Priority:** All newly arriving processes should start at the Medium priority level.

(b) **Priority Updates:** When a process reappears, its priority should be revised according to its average burst time recorded so far, and it should be placed in the highest queue for which average burst time is less than allotted time slice. Only consider the process which didn't end with Error.

2. Shortest Job First (SJF)

(a) **Initial Burst Time:** For a new process with no prior history, assign a default burst time of 1 second for its very first execution.

(b) **Historical Burst data:** For subsequent executions, estimate the burst time as the average of up to the last k valid executions. That is, for the n^{th} execution, use the mean of $\min(n-1, k)$ most recent valid burst times. Executions that ended in error are excluded from this history.

Input for FCFS:

```
cmd1, cmd2, cmd2, cmd1 are all inputted at the start with no delay
```

Expected Behaviour:

```
Execute cmd1 considering default burst time 1 second.
Records burst time of cmd1, say 1sec.
Execute cmd2 considering default burst time 1 second.
Records burst time of cmd2, say 2sec.
Now run cmd1 before cmd2 because it's burst time < cmd2's.
Execute cmd2.
```

Assumptions

- Assume the time when your scheduler starts to be t=0.
- Burst time >>> quantum.
- Length of each command is less than 1000.
- Number of commands in real time is less than 50

Output Format for both Parts

- After each context switch, please print the following details each on a new line:

$$\langle Command \rangle, \langle Start\_Time \rangle, \langle End\_Time \rangle$$

- After termination of a process print a single line of the following details: Task: For each Task Scheduler, print the following process details into a CSV file named 'result_type_acronym.csv' (e.g. result_offine_RR.csv). The CSV file should contain the following columns in order (Do not include headers in the csv):

1. Command - The command associated with the process.
2. Finished - Indicates whether the process has finished execution (‘Yes‘/‘No‘).
3. Error - Indicates whether an error occurred during the process execution (‘Yes‘/‘No‘).
4. Completion Time in milliseconds (uint64_t)
5. Turnaround Time in milliseconds (uint64_t)
6. Waiting Time in milliseconds (uint64_t)
7. Response Time in milliseconds (uint64_t)

We will not check the entries in the time-related sections if an error occurs in the command, you can fill these with any dummy values.

Submission

- Submit a C header file named ’offline_schedulers.h’ and ‘online_schedulers.h’ contained within a zipped folder named `<Entry_Number>`. Only include these two files in the submission.
- Ensure your code is well-commented and follows best practices for readability and maintainability.
- All submissions will be strictly checked for plagiarism, including comparisons with submissions from previous years as well as AI-generated content. Any instance of plagiarism will result in a score of zero for the assignment.

Good luck, and happy coding!