Implementation of Scheduling Algorithms in C

Objective:

The purpose of this assignment is to deepen understanding of CPU scheduling algorithms through implementation.

Tasks:

Task 1: Offline Scheduling

The list of processes will be provided as input before the program starts. Refer to offline_schedulers.h. Implement the following CPU scheduling algorithms in C:

1. First-Come, First-Served (FCFS)
2. Round Robin (RR)
3. Multi-level Feedback Queue (MLFQ), with three Queues:
    Q0: High priority
    Q1: Medium priority
    Q2: Low priority

Example:
FCFS:
Say the process array is as follows,
Process {Command – 'run p1', Command – 'run p2', Command – 'run p3'},
where run p1, run p2, run p3 etc. are dummy processes running for 1, 2, 3 sec respectively.
Then your FCFS scheduler should

1. Execute: run p1, context switch when it terminates print the results as expected.
2. Execute: run p2, context switch when it terminates print the results as expected.
3. Execute: run p2, context switch when it terminates print the results as expected.

And terminate the scheduler on completion of all processes.

*run p1, run p2, run p3 are dummy processes.

Task 2: Online Scheduling

In this task, processes should be taken as input in real-time. After each context switch, the terminal input must be checked to receive new processes (any new command will be deemed new process if it is newline('\n') terminated). Refer to online_schedulers.h. Implement the following algorithms:

- Multi-level Feedback Queue (MLFQ) with Adaptive Features
    - Initial Priority Assignment: Assign a **Medium** priority to each process when it first arrives.

    - Priority Adjustment: Update the process's priority based on the average burst time when it appears next. Only consider the process which didn't end with Error as historical data.
- Shortest Job First (SJF)
    - Initial Burst Time Assignment: When a process first appears, assign it a default burst time of 1 second, as the actual burst time is unknown.
    - Historical Data: Update the burst time for a process using historical data when it reappears and continue updating it as more (to average burst time) as more data becomes available. Only consider the process which didn't end with Error as historical data.
- Shortest Remaining Time First (SRTF): Similar to Shortest Job First

Example:

SJF:

Start the scheduler input will be given on the terminal to be taken by standard input until a new line is received. When a new line is received the preceding string should be deemed as a command. Say we entered p1, p2, p2, p1 in order at the start simultaneously without delay.

Then

1. Execute p1 considering default burst time 1 second. Records burst time say 1sec.
2. Execute p2 considering default burst time 1 second. Records burst time say 2sec.
3. Now run p1 before p2 because it's burst time < p2's.
4. Execute p2.

*p1, p2 are dummy processes.

OUPTUT:

- After each context switch, please print the following details each on a new line:
  <Command>|<Start Time of the context>|<End Time of the context>

- 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 for scheduler>.csv` (e.g. result_offine_RR.csv).
  The CSV file should contain the following columns:

  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)

Submission Requirements:

Source Code:
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.

Evaluation Criteria:

Correctness of Implementation:
The algorithms must function correctly as per the scheduling logic.
Code Clarity and Readability:
Code should be well-documented, formatted, and easy to understand.