

CS1234 - L3: Task Scheduler

1. Objective

- In an Operating System, the **Scheduler** is responsible for deciding the order in which tasks are executed.
- Your goal is to simulate a simple scheduler by reading a list of tasks and organizing them based on a specific scheduling scheme.
- You must only use a **linked list** for storing and manipulating the tasks.
- Begin your implementation from the starter code in **L3.c**, and rename the file to your roll number **CS25Bxxx.txt** before submitting (Moodle is not accepting C file submissions).

2. Input Format

1. **First line:** **N** (total number of tasks).
2. **Second line:** **<Scheme>** (scheduling scheme to use: either **FCFS** or **SJF**).
3. **Next N lines:** **<TaskID> <Duration>**
 - **TaskID:** A string in the format **Txxx**.
 - **Duration:** An integer representing the time required for the task.

Constraints: $1 \leq N \leq 1000$, $1 \leq \text{Duration} \leq 10000$

Sample Input:

```
3
SJF
T001 10
T002 5
T003 5
```

3. Schemes

FCFS (First Come First Served)

- Tasks are executed in the exact order they arrive.
- Define an **insertTail()** function. Every new task is added to the end of the list.

SJF (Shortest Job First)

- Tasks with the shortest duration are executed first.
- Define an **insertSorted()** function. As you read the input, insert the task into the correct position in the list so that the list remains sorted by duration.
- Tie-breaker: If two tasks have the same duration, the one that appeared earlier in the input should come first.

4. Computation

1. **Order:** The sequence of Task IDs separated by a space.
 - This can be computed by traversing the linked list.
 - Remember to free the dynamically allocated memory after processing the list.
2. **Average waiting time:**
 - Waiting time of a task is the sum of durations of all previous tasks.
 - Calculate the average as the sum of waiting times divided by N.
 - Print the average rounded to two decimal places, as given below.
`printf("%.2f", roundTwo(totalWaitTime / N));`

5. Output Format

1. **First Line:** `<TaskID1> <TaskID2> ... <TaskIDN>`
2. **Second Line:** `<Average_Waiting_Time>`

Sample Output:

```
T002 T003 T001
5.00
```

Explanation:

- **Order:** T002 (5ms) -> T003 (5ms) -> T001 (10ms).
- **T002:** Starts immediately. Wait = 0.
- **T003:** Starts after T002 finishes. Wait = 5.
- **T001:** Starts after T002 + T003 finish (5+5). Wait = 10.
- **Average waiting time:** 5.00 ms.

6. Evaluation

The assignment includes 10 public test cases in the `tests/` folder. Files named `ip_xx` contain the inputs, and the corresponding `ep_xx` files contain the expected outputs. Each test case carries 0.5 marks.

Testing in interactive mode:

- Compile the code using `gcc` and link the math library with the `-lm` flag.
`gcc solution.c -lm -o solution.out`
- Execute the output file and provide the inputs as specified in the problem statement.
`./solution.out`

Validating all public test cases:

- Execute the evaluator script with the filename passed as an argument.
`./evaluator.sh solution.c`

7. Practice Problems

Simulate the following additional scheduling schemes using the same linked list approach.

RR (Round Robin): Execute processes in a circular loop using a fixed time quantum.

- Process the **head** node.
- Run for $\min(\text{Remaining_Time}, \text{Time_Quantum})$.
- Update **Remaining_Time** by subtracting the time run.
- Update the global timer in **Current_Time**.
- If **Remaining_Time** > 0: Move the **head** node to the **tail**.
- If **Remaining_Time** == 0: Remove **head** and record the completion time.
- Repeat till all tasks have been completed.

HRRN (Highest Response Ratio Next): Select the task with the highest RR next.

$\text{Response Ratio} = (\text{Waiting Time} + \text{Task Duration}) / \text{Task Duration}$

- Maintain a ready queue of arrived but unfinished processes.
- For each process in the queue, compute its RR using the current waiting time.
- Select the process with the highest response ratio.
- Execute the selected process until completion.
- Update the waiting times of the remaining processes.
- Repeat until all processes are completed.