CS29003 ALGORITHMS LABORATORY
ASSIGNMENT 12
Date: 31st Oct, 2019

Important Instructions

1. **List of Input Files:** Input.txt
2. **Output Files to be submitted:** ROLLNO_A12_P1.c/.cpp, ROLLNO_A12_P2.c/.cpp
3. **THERE WILL BE PART MARKING.**
4. Files must be read using file handling APIs of C/C++. Reading through input redirection isn't allowed.
5. You are to stick to the file input output formats strictly as per the instructions above. Failing to do so might cause you to lose marks.
6. Write your name and roll number at the beginning of your program.
7. Do not use any global variable unless you are explicitly instructed so.
8. Use proper indentation in your code.
9. Submission through **.zip files aren't allowed**.

Priority based Shortest Remaining Time First (SRTF) Scheduling

Intro

CPU Scheduling is a process which allows one process to use the CPU while the execution of another process is on hold (in waiting state) due to unavailability of any resource like I/O etc, thereby making full use of CPU. The aim of CPU scheduling is to make the system efficient, fast and fair.
There are basically two types of scheduling: Preemptive and Non-Preemptive Scheduling. Under the non-preemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until the process has finished it's execution. In the preemptive scheduling, the process are given execution time based on their priority. In this scheduling, the running task is interrupted for some time if a higher priority process is available, and then resumed later.
The Shortest Remaining Time First (SRTF) Scheduling is a Preemptive scheduling algorithm where every process is assigned a priority. The process with the highest priority is executed first. In case multiple processes have the same priority, the one with the lowest remaining CPU time is selected for execution. As this is a preemptive algorithm, if a higher priority process enters the system, the current process is stopped and the CPU is assigned to the higher priority process.

Objectives of this assignment

In this assignment, you are required to create the SRTF scheduling algorithm that processes the jobs based on their priority. You would need to prepare a priority queue using heap data structure. In this case, a job can have priority from 1 to 5, with 1 being the highest priority. The assignment has two parts. In part I, you would need to create the Heap data structure and other necessary operations to implement the SRTF Scheduling algorithm. Your code will read a list of processes from a file and schedule them based on the SRTF algorithm. In part II, you will have to add the *Ageing* technique to your program. This will ensure that processes that have not been executed for a long time are assigned higher priority, and eventually get CPU time.

Part I - Priority Queue

As a first step, you would need to store the processes based on the priority assigned. In this case, the priority of a process can vary from 1 to 5. This could be accomplished by using a heap data structure, specifically a min heap. The entries in the min heap would be based on the remaining CPU time of the process. Thus the root node will represent the process with the least remaining CPU time.
You would need to maintain 5 heaps for each priority. Each process would be inserted into the corresponding heap based on its priority. To select a process for CPU execution, you will search for a process from the highest priority heap to the lowest. Thus, you will start from priority heap 1, then priority heap 2, then priority heap 3 and so on.
Remember that the scheduling is preemeptive in nature. After every unit of system clock, you will need to check if a higher priority process has arrived. If there is a new higher priority process (or same priority but smaller remaining time), you would need to stop the current process and start the execution of the higher priority process.
The Heap data structure should have at least these elements:

1. **job ID:** a unique number for each process
2. **process Burst Time:** the total amount of CPU time required for this process to complete.
3. **process Remaining Time:** the amount of CPU time **remaining** for this process to complete.
4. **process Arrival Time:** the time at which the process was submitted to the system.
5. **process Priority:** the priority assigned to the process. The value can range from 1-5, with 1 being the highest.

It should also support at least these operations:

1. **getMin():** Return the root element of the Min Heap.
2. **extractMin():** Remove the minimum element from Min Heap.
3. **insert():** Insert a new element into the Heap.

Note:

1. **Tie Break:** In case multiple processes have the same priority, the process with the lowest remaining burst time will be chosen. In case there is still a tie, the process with the lowest job ID is chosen. The order is : process priority $>$ process remaining time $>$ job ID
2. **Order in min heap:** For each min heap, the process should be sorted based on the remaining CPU time required.
3. **System clock:** You will need to maintain an integer counter starting from zero which will act as the system clock.

Consider the example of scheduling given in Figures 1, 3 and 4. The system input will consist of Job ID, Arrival time, Burst time, and Priority. The Job ID is a unique integer which represents the process. The arrival time represents the time at which the process will be given to the system. This time represent the system time. The Burst time represent the amount of system time required for this process to complete its execution. The priority represents the priority assigned to the process.

Part II - Process Ageing

The priority based scheduling technique could introduce a problem known as *Starvation*. In this problem, a process would have to wait indefinitely for the CPU because of its lower priority. Ageing is used to ensure that processes with lower priority will eventually complete their execution. This technique can be used to reduce starvation of low priority tasks. The basic idea is to increase the priority of a process if it has not received CPU time for a long time.
You are required to implement an ageing mechanism where you will check the status of all the process in every 20 units of system clock and increase the priority of all the process which have not received CPU in the last 20 units. Thus, a process which has priority 3, and has not received any CPU time in the last 20 units will be give a priority of 2.

Input

Job ID	1	2	3	4	5	6
Arrival Time	0	0	0	0	0	0
Burst Time	10	1	2	1	5	4
Priority	3	1	4	5	2	2

Output

P1	10	20
P2	0	1
P3	20	22
P4	22	23
P5	5	10
P6	1	5

Figure 1: Scheduling with same arrival time but different priority

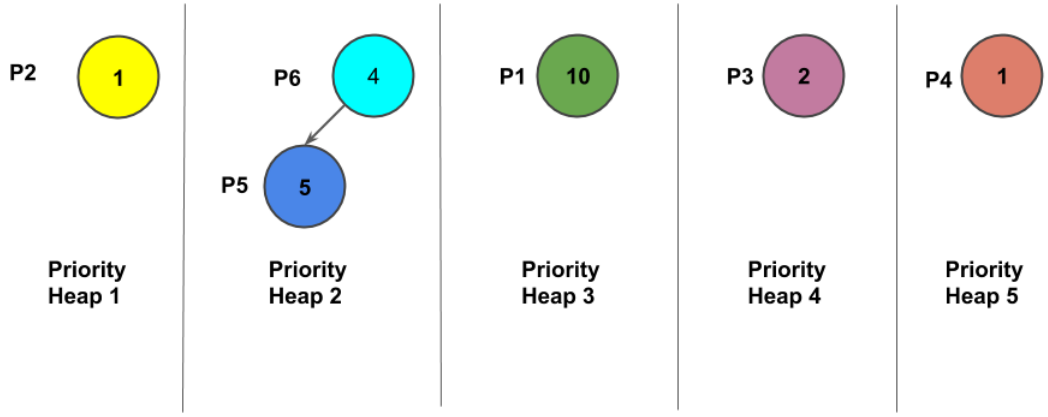
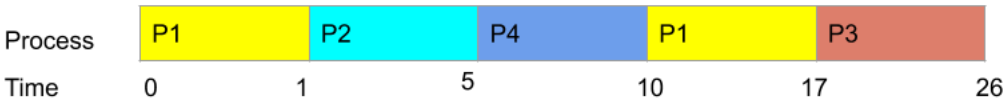


Figure 2: Five priority heaps for the processes in Figure 1.

Input

Job ID	1	2	3	4
Arrival Time	0	1	2	3
Burst Time	8	4	9	5
Priority	1	1	1	1



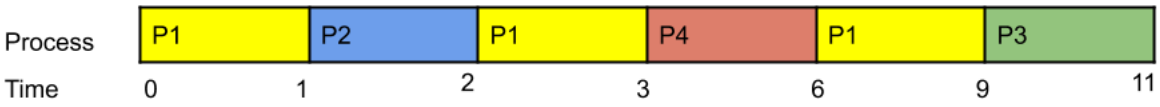
Output

P1	0	17
P2	1	5
P3	17	26
P4	5	10

Figure 3: Scheduling with same priority but different arrival time

Input

Job ID	1	2	3	4
Arrival Time	0	1	2	3
Burst Time	5	1	2	3
Priority	3	1	4	2



Output

P1	0	9
P2	1	2
P3	9	11
P4	3	6

Figure 4: Scheduling with different priority and arrival time

Note:
The code for ageing will run once every 20 units of system time and it should run after the SRTF scheduling code.

```
// system_time represents the system time
While(True)
{
    cpu_execution(heap_structure);
    if system_time%20==0:
    {
        check_and_increase_priority(heap_structure);
    }

    system_time ++;
}
```

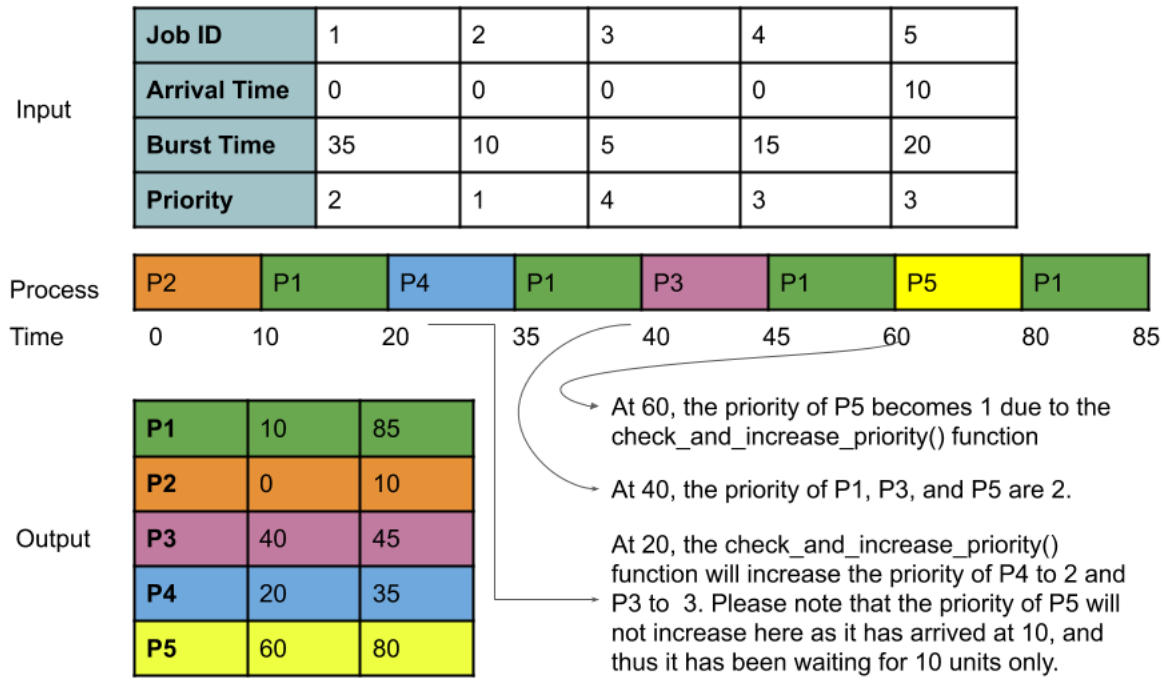


Figure 5: Scheduling with ageing

Input and Output Format

The first line of the input is an integer (say N). N indicates number of processes for which the details are provided in the subsequent N lines. Each such line contains the job id of one process, the corresponding arrival time, burst time and its priority.
The output will contain N lines, each of which will indicate the job id of a process, the corresponding start and end time in the scheduling. The output lines should be sorted in ascending order of their job ids.

FILE: *Input.txt*

```
6
1 0 10 3
2 0 1 1
3 0 2 4
4 0 1 5
5 0 5 2
6 0 4 2
```

FILE: *Output.txt*

```
1 10 20
2 0 1
3 20 22
4 22 23
5 5 10
6 1 5
```

Note: The output above indicates the results from part I. The output from part II should follow the same format.