WORK SHEET 4

Part A: Longest Job First

Longest Job First (LJF) is a non-preemptive scheduling algorithm. This algorithm is based on the burst time of the processes. The processes are put into the ready queue based on their burst times i.e., in descending order of the burst times. As the name suggests this algorithm is based on the fact that the process with the largest burst time is processed first. The burst time of only those processes is considered that have arrived in the system until that time. Its preemptive version is called Longest Remaining Time First (LRTF) algorithm.

Characteristics of Longest Job First(Non-Preemptive)

- Among all the processes waiting in a waiting queue, the CPU is always assigned to the process having the largest burst time.
- If two processes have the same burst time then the tie is broken using **FCFS** i.e. the process that arrived first is processed first.
- LJF CPU Scheduling can be of both preemptive and non-preemptive types.

Advantages of Longest Job First(LJF)

- No other process can execute until the longest job or process executes completely.
- All the jobs or processes finish at the same time approximately.

Disadvantages of Longest Job First CPU Scheduling Algorithm

- This algorithm gives a very high **average waiting time** and **average turn-around time** for a given set of processes.
- This may lead to a convoy effect.
- It may happen that a short process may never get executed and the system keeps on executing the longer processes.
- It reduces the processing speed and thus reduces the efficiency and utilization of the system.

Longest Job First CPU Scheduling Algorithm

- Step-1: First, sort the processes in increasing order of their Arrival Time.
- Step 2: Choose the process having the highest Burst Time among all the processes that have arrived till that time.
- Step 3: Then process it for its burst time. Check if any other process arrives until this process completes execution.
- Step 4: Repeat the above three steps until all the processes are executed.

Part B: Longest Remaining Time First

Longest Remaining Time First (LRTF) is a **preemptive** version of **Longest Job First (LJF) scheduling algorithm**. In this scheduling algorithm, we find the process with the maximum remaining time and then process it, i.e. check for the maximum remaining time after some interval of time(say 1 unit each) to check if another process having more **Burst Time** arrived up to that time.

<u>Characteristics of Longest Remaining Time First (LRTF)</u>

•Among all the processes waiting in a waiting queue, CPU is always assigned to the process having largest burst time.Characteristics of Longest Remaining Time First (LRTF)

•If two processes have the same burst time then the tie is broken using **FCFS** i.e. the processthat arrived first is processed first

•LJF CPU Scheduling can be of both preemptive and non-preemptive type.

<u>Advantages of Longest Remaining Time First (LRTF)</u>

•No other process can execute until the longest job or process executes completely.

•All the jobs or processes finishes at the same time approximately.

<u>Disadvantages of Longest Remaining Time First (LRTF)</u>

•This algorithm gives very high **average waiting time** and **average turn-around time** for a given set of processes.

•This may lead to convoy effect.

•It may happen that a short process may never get executed and the system keeps on executing the longer processes.

•It reduces the processing speed and thus reduces the efficiency and utilization of the system.

Write a C/C++ code for LRTF CPU Scheduling algorithm Where We have given some processes with arrival time and Burst Time and we have to find the completion time (CT), Turn Around Time(TAT), Average Turn Around Time (Avg TAT), Waiting Time(WT), Average Waiting Time (AWT) for the given processes. **LRTF is a preemptive scheduling algorithm. Its tie-breaker is FCFS and if FCFS does not breaks the tie then, we use process id as the tie-breaker.** Since completion time (CT) can be directly determined by Gantt chart, and

Turn Around Time (TAT) = (Completion Time) - (Arrival Time)

Also, Waiting Time (WT) = (Turn Around Time) - (Burst Time)

Part C: Higest Response Ratio Next

Given N processes with their **Arrival times and Burst times**, the task is to find average waiting time and an average turn around time using HRRN scheduling algorithm.
The name itself states that we need to find the response ratio of all available processes and select the one with the highest Response Ratio. A process once selected will run till completion.

Characteristics of HRRN CPU Scheduling:

- Highest Response Ratio Next is a non-preemptive CPU Scheduling algorithm and it is considered as one of the most optimal scheduling algorithm.
- The criteria for HRRN is Response Ratio, and the mode is Non-Preemptive.
- HRRN is basically considered as the modification of **Shortest Job First** in order to reduce the problem of **starvation**.
- In comparison with SJF, during HRRN scheduling algorithm, the CPU is allotted to the next process which has the highest response ratio and not to the process having less burst time.

Response Ratio = (W + S)/S

Here, W is the waiting time of the process so far and S is the Burst time of the process.

Advantages of HRRN CPU Scheduling

- HRRN Scheduling algorithm generally gives better performance than the **shortest job first** Scheduling.
- There is a reduction in waiting time for longer jobs and also it encourages shorter jobs.

Disadvantages of HRRN CPU Scheduling

- The on ground implementation of HRRN scheduling is not possible as it is not possible know the burst time of every job in advance.
- In this scheduling, there may occur overload on the CPU.

Performance of HRRN –

- Shorter Processes are favoured.
- Aging without service increases ratio, longer jobs can get past shorter jobs.

Part D: Preemptive Priority CPU Scheduling

Preemptive Priority CPU Scheduling Algorithm is a pre-emptive method of **CPU scheduling algorithm** that works based on the priority of a process. In this algorithm, the scheduler schedules the tasks to work as per the priority, which means that a higher priority process should be executed first. In case of any conflict, i.e., when there is more than one process with equal priorities, then the pre-emptive priority CPU scheduling algorithm works on the basis of **FCFS (First Come First Serve) algorithm**.

How does Preemptive Priority CPU Scheduling Algorithm decide the Priority of a Process?

Preemptive Priority CPU Scheduling Algorithm uses a rank-based system to define a rank for each process, where lower rank processes have higher priority and higher rank processes have lower priority. For instance, if there are 10 processes to be executed using this Preemptive Algorithm, then process with rank 1 will have the highest priority, the process with rank 2 will have comparatively lesser priority, and process with rank 10 will have least priority.

How does Preemptive Priority CPU Scheduling Algorithm work?

- Step-1: Select the first process whose **arrival time** will be 0, we need to select that process because that process is only executing at time t=0.
- Step-2: Check the priority of the next available process. Here we need to check for 3 conditions.
    - if $\text{priority}_{(\text{current\_process})} > \text{priority}_{(\text{prior\_process})}$:- then execute the current process.
    - if $\text{priority}_{(\text{current\_process})} < \text{priority}_{(\text{prior\_process})}$:- then execute the prior process.
    - if $\text{priority}_{(\text{current\_process})} = \text{priority}_{(\text{prior\_process})}$:- then execute the process which arrives first i.e., arrival time should be first.
- Step-3: Repeat Step-2 until it reaches the final process.
- Step-4: When it reaches the final process, choose the process which is having the highest priority & execute it. Repeat the same step until all processes complete their execution.

Implementing priority CPU scheduling. In this problem, we are using **Min Heap** as the data structure for implementing priority scheduling.

In this problem smaller numbers denote higher priority.

The following functions are used in the given code below:

```
struct process {
   processID,
   burst time,
   response time,
   priority,
   arrival time.
}
```

•**void quicksort(process array[], low, high)**: This function is used to arrange the processes in ascending order according to their arrival time.

•**int partition(process array[], int low, int high)**: This function is used to partition the array for sorting.

•**void insert(process Heap[], process value, int *heapsize, int *currentTime)**: It is used to include all the valid and eligible processes in the heap for execution. heapsize defines the number of processes in execution depending on the current time currentTime keeps record of the current CPU time.

•**void order(process Heap[], int *heapsize, int start)**: It is used to reorder the heap according to priority if the processes after insertion of new process.

•**void extractminimum(process Heap[], int *heapsize, int *currentTime)**: This function is used to find the process with highest priority from the heap. It also reorders the heap after extracting the highest priority process.

•**void scheduling(process Heap[], process array[], int n, int *heapsize, int *currentTime)**: This function is responsible for executing the highest priority extracted from Heap[].

•**void process(process array[], int n)**: This function is responsible for managing the entire execution of the processes as they arrive in the CPU according to their arrival time.