

Assignment # 2

**Submission:** You are required to submit a single header file for the assignment. The **header file should contain all the code**. Name the header files as *ProcessScheduler.h* and submit it on the slate.

**Deadline:** Deadline to submit the assignment is **31th October 2:00 PM**. No submission will be considered for grading outside the slate or after 31th October 2:00 PM. **Correct and timely submission of the assignment is the responsibility of the student; hence no relaxation will be given to anyone.**

**Plagiarism:** -50% marks in the assignment if any part of the assignment is found plagiarized.

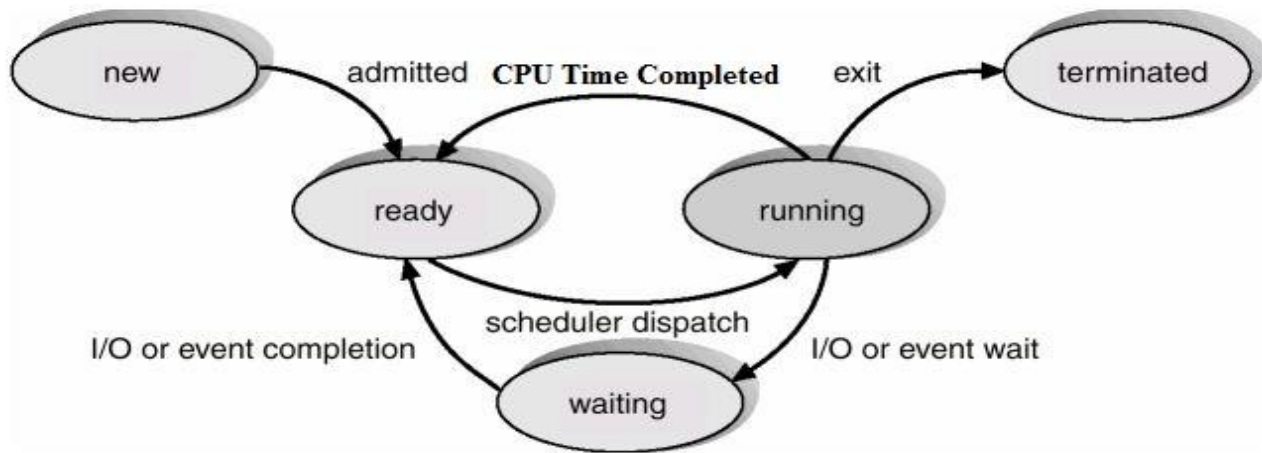
## Assignment Description:

### Process Concept

A program is passive while process is active. Attributes held by a process include: CPU, Memory and Hardware state.

Process has five basic states:

1. **New:** Process is created and stored in **Process List**.
2. **Ready:** The process can be executed anytime if CPU is assigned. In other words process is not waiting for the completion of any I/O operations.
3. **Running:** Instructions of the process are being executed. A single CPU can execute one process at any time. Therefore only one process can be in running state.
4. **Waiting:** Waiting for the completion of an I/O operation, e.g., waiting to get user input from keyboard, writing/reading to a file. The completion of an I/O operation is indicated by an I/O event.
5. **Terminated:** The process is finished and its memory and resources are deallocated.



**Figure 1: Process Scheduling**

We have to simulate Process Scheduling as shown in **Figure 1** that will be implemented using Multilevel Feedback Queues & Waiting Queues using Singly Linked List. In order to simulate, we need to maintain queue for **ready state** and **waiting state**. Because multiple processes can be in ready and waiting state.

## CPU

There is only one CPU. This will run the single instruction at a time. To avoid a single process monopolizing the system resources and always using CPU. The CPU time is divided or sliced between processes. A given process will get CPU for a certain system defined **Time Slice**. Once the time slice is finished CPU is assigned to another process from the **ready queue**.

If a process issues an I/O operation during its execution by CPU. The process has to wait for the completion of the I/O operation. Therefore, process is added to the **waiting queue** and CPU is assigned to another process from the **ready queue**.

## Time in ticks

For simulation we will assume a global time starting from 0. Every time CPU executes one instruction of a process, the **global time** is incremented by 1. Even the I/O instructions are first executed by CPU before the process is blocked and these instructions count towards global time.

Note: For this simulation, we assume the time to move process from one state to another state is negligible and not counted towards the global time.

## Process List

When a process enters a system it is added to a doubly linked list of processes. Each process irrespective of its state (i.e., ready, block ...) has an entry into the doubly linked list sorted according to their arrival time. By arrival means a new process is created and added to **process list** (doubly linked list). This list has complete information about processes (i.e. Process Name, State, Arrival Time, Instructions, etc.).

Note: The ready and waiting queues will only contain the **index** of the processes of **Process List**.

**General structure of process contains:**

1. Process Name.
2. Index of Process in Process List.
3. State i.e. Ready, Running, Waiting and Terminated.
4. Arrival Time (process arrival time)
5. List of instructions.
6. Current Queue (Ready queue: FCFS-1, FCFS-2 or SJF. Waiting Queue: HardDisk, Network or Device (Peripheral Device) )

## **Ready Queues**

### **FCFS (First Come First Serve)**

It is a traditional scheduling technique in which all the processes have same priority, FCFS scheduled the processes in the order in which they come, process which come first will take CPU time slice first then next and so on, processes are inserted into the Rear of a queue when they are submitted.

### **SJF (Shortest Job First)**

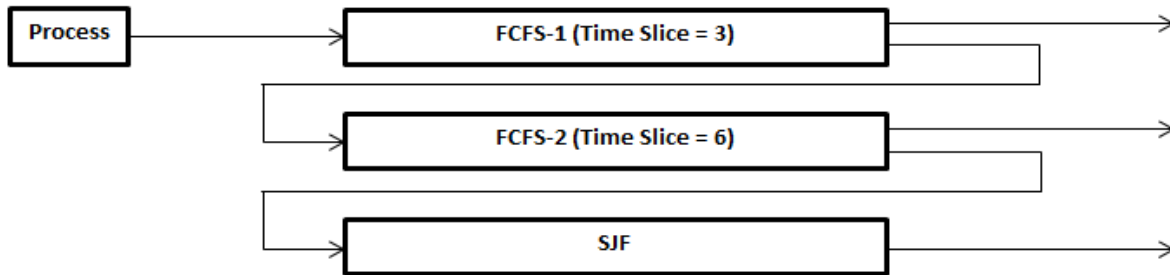
The process which has short burst time will schedule first to run then next shortest job (process) gets the CPU time. A scheduler arranges the processes with the least burst time in Front of the queue and longest burst time in Rear of the queue. Process has to be executed to its entirety (except if I/O event occurred).

Note: **Burst time**: Requires advanced knowledge or estimations about the time required for a process to complete. So you should keep track of how many numbers of instructions left. Number of instructions will tell us about burst time.

## **Multilevel Feedback Queues**

In MLFQ (Multi-level feedback queue) scheduling, the queue (MLFQ) is divided into three queues, where two queues have FCFS scheduling technique and the remaining one has SJF scheduling technique.

1. FCFS scheduling technique with highest priority (FCFS-1).
2. FCFS scheduling technique with medium priority (FCFS-2).
3. SJF scheduling technique with lowest priority.



**Figure 2: Ready Queues (MLFQ)**

## Scheduling

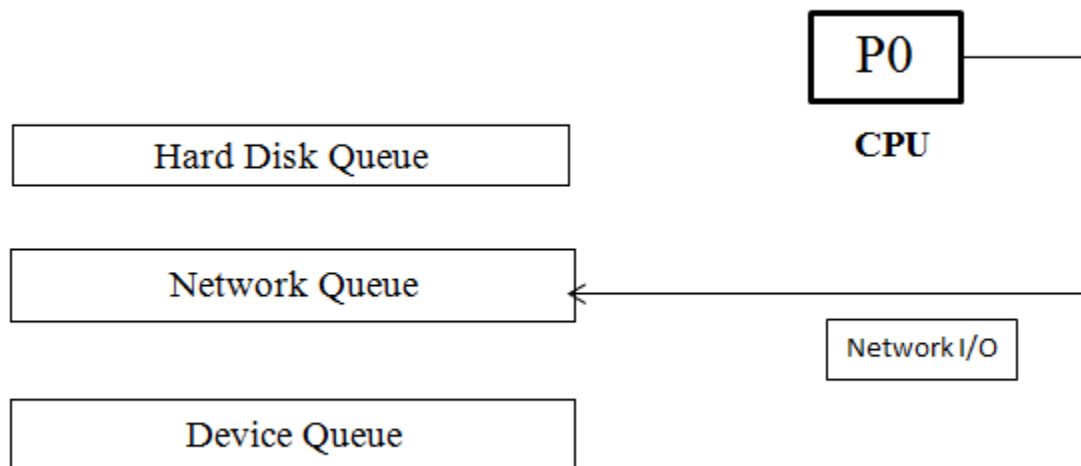
1. A new process is inserted at the Rear of highest priority queue (FCFS-1) according to the arrival time.
2. At some stage the process reaches the Front of the queue and is assigned the CPU.
3. If the process uses the entire time slice, it is switched and inserted at the Rear of the next lower level queue. This next lower level queue will have a time slice which is more than that of the previous higher level queue.
4. This scheme will continue until the process completes or it reaches the base level queue (SJF).
5. SJF will sort (ascending order) the process whenever process enter in it on the basis of instructions left. And if above higher priority queues are empty then it runs the process till it completes.
6. If the I/O event occurred, process will be inserted at the Rear of waiting queue, and when the process becomes ready again it is inserted in the highest priority queue. (see **Waiting Queue**)
7. For scheduling, the scheduler always starts picking up processes from the Front of the highest level queue. If the highest level queue has become empty, then only will the scheduler take up a process from the next lower level queue. The same policy is implemented for picking up in the subsequent lower level queues. Meanwhile, if a process comes into any

of the higher level queues, it will switch to that higher level queue after completing its time slice.

Also, a new process is always inserted at the highest queue with the assumption that it would be a short time consuming process. Long processes will automatically sink to lower level queues based on their time consumption and interactivity level. In the multilevel feedback queue, a process is given just one chance to complete at a given queue level before it is forced down to a lower level queue.

## Waiting Queue

There will be three waiting queues. All of them use FCFS scheduling technique having same priority. Whenever CPU encounters I/O operations Hard Disk, Network & Peripheral devices (i.e. Monitor, Keyboard and Printer) that specific process will move to their respective queue as shown in Figure 3. And wait for specified time. When I/O operation is completed, process will move back to highest priority ready queue.



**Figure 3: Waiting Queues (Instruction Read Network 2 ticks)**

Waiting queues are independent to each other. But within the queue, processes will wait for the prior process to complete its I/O operation.

## Instructions of Process

The data between **Start** and **End** of the input file specify a set of instructions associated with each process. There are three types of instructions, **READ**, **WRITE** and **COMPUTE**. These instructions

represent CPU computations and read/write data to the Hard Disk, Network and Peripheral devices (Monitor, Keyboard and Printer). **COMPUTE** instruction always will be 1 tick. Different keywords as below:

**DISK:** This keyword represents the hard disk.

**SCREEN:** represents the monitor. And it's a Peripheral device.

**PRINTER:** represents the printer. And it's a Peripheral device.

**KEYBOARD:** represents the keyboard. And it's a Peripheral device.

**NETWORK:** represents the Network communication.

Example:

READ KEYBOARD 1 Tick

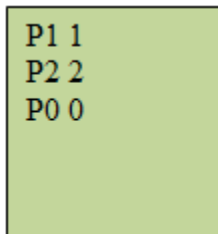
Read from keyboard while waiting time is 1 tick.

## Description of Files:

Following files will be provided for Process Scheduling

### 1. job.txt

This file will contain process name and their arrival time separated with space. i.e.



```
P1 1
P2 2
P0 0
```

Note: List of processes is unordered.

### 2. ProcessName.txt

a) Examples of this file are P0.txt , P1.txt and P2.txt

P0.txt

```
Start  
COMPUTE  
COMPUTE  
COMPUTE  
READ SCREEN 2 tick  
COMPUTE  
COMPUTE  
COMPUTE  
End
```

P1.txt

```
Start  
COMPUTE  
READ SCREEN 2 tick  
WRITE NETWORK 3 tick  
COMPUTE  
WRITE DISK 2 tick  
End
```

P2.txt

```
Start  
COMPUTE  
COMPUTE  
COMPUTE  
COMPUTE  
COMPUTE  
COMPUTE  
COMPUTE  
COMPUTE  
COMPUTE  
COMPUTE  
COMPUTE  
COMPUTE  
COMPUTE  
End
```

### Function for testing:

```
void runProcessScheduling(const char* PathofJobFile, int globalTick)
```

1. const char\* PathofJobFile :- path of job.txt file.
2. int globalTick :- simulation can be specified to run for a certain global ticks (ticks vs. time)
3. Path for ProcessName.txt will be same as job.txt.

After every tick of the global clock, you have to write data to files

#### 1) **Process.txt**

Name of process (in order of their arrival time), state, which queue process resides OR CPU

#### 2) **CPU.txt**



Name of process currently being executed, instruction being executed.

3) **Queue.txt**

Name of queue, list of processes in the queue.