**OPERATING SYSTEM**

**CSE-316**

**PROJECT REPORT**



**Topic-** Multiplelevel-Queue-Scheduling-Algorithm

# Submitted By: Submitted To:

Raj Cherry Khosla

K22UP Rajveer UID: 13436

R.No:53

**QUESTION**

Write a program for multilevel queue scheduling algorithm. There must be three queues generated. There must be specific range of priority associated with every queue. Now prompt the user to enter number of processes along with their priority and burst time. Each process must occupy the respective queue with specific priority range according to its priority. Apply Round Robin algorithm with quantum time 4 on queue with highest priority range. Apply priority scheduling algorithm on the queue with medium range of priority and First come first serve algorithm on the queue with lowest range of priority. Each and every queue should get a quantum time of 10 seconds. CPU will keep on shifting between queues after every 10 seconds.

**INTRODUCTION**

Multilevel queue scheduling is a popular process scheduling algorithm used in operating systems to manage processes based on their priority. In this scheduling algorithm, processes are divided into multiple priority queues, each with its own priority range. Each queue is then scheduled using a different scheduling algorithm, often based on the characteristics of the processes in that queue. Multilevel queue scheduling allows for better handling of different types of processes and improves overall system efficiency.

**Program Description:**

The provided Python program demonstrates a multilevel queue scheduling algorithm with three priority queues. Each queue has a specific priority range, and the program prompts the user to input the number of processes, their priorities, and burst times. The program then schedules the processes into the appropriate queues and applies different scheduling algorithms for each queue:

**Highest Priority Queue (Queue 1)**: Round Robin scheduling is applied with a time quantum of 4 seconds.

**Medium Priority Queue (Queue 2)**: Priority Scheduling is used to select processes for execution, prioritizing processes with lower numerical priority values.

**Lowest Priority Queue (Queue 3):** First Come First Serve (FCFS) scheduling is applied, where processes are executed in the order they arrive.

The program simulates the execution of processes in each queue, shifting the CPU's focus between queues every 10 seconds to give each queue a fair share of CPU time.

**Requirement of the solution**

**1. Data Structures:**

The program uses two struct definitions: Process and Queue. These structures help store information about individual processes and organize processes into priority queues.

**2. isNotComplete Function:**

This function checks whether any process in any of the queues has remaining execution time. It iterates through the queues and processes within them, checking the remaining\_time of each process. If any process has remaining time, it sets notComplete to true, indicating that the scheduling needs to continue.

**3. sortProcessesByPriority Function:**

This function sorts the processes in a given queue by their priorities in descending order. This is essential for the medium-priority queue, which uses a Priority Scheduling algorithm where lower priority values are favored.

**4. updateTotalTime Function:**

The updateTotalTime function iterates through all the queues, processes, and increments the turnaround\_time for each process that has remaining execution time. It also increments the total\_time for each queue that has not been fully executed.

**5. Main Function:**

The main function initializes three priority queues (queues) based on the specified priority ranges.

It prompts the user to input the number of processes and their priorities and burst times.

Processes are assigned to their respective queues based on priority.

Memory is allocated for processes in each queue, and processes are assigned to their corresponding queues.

The program then simulates process execution within the queues using different scheduling algorithms based on the current queue.

Messages are printed to indicate which queue is currently being executed, which process is running, and the time spent on each process.

The program calculates and displays the turnaround time and waiting time for each process and provides average turnaround and waiting times.

**6. Explanation of Execution:**

The program simulates the execution of processes in each queue. The specific logic for each queue is as follows:

Queue 1 (Highest Priority): Round Robin scheduling with a quantum of 4 seconds.

Queue 2 (Medium Priority): Priority Scheduling with processes sorted by priority in descending order.

Queue 3 (Lowest Priority): First Come First Serve (FCFS) scheduling.

The program keeps track of time spent and the current queue to ensure that each queue receives a fair share of CPU time.

**7. Results:**

The program calculates and displays the turnaround and waiting times for each process in each queue.

It provides the average turnaround time and average waiting time for all processes.

***GITHUB LINK:***

***<https://github.com/Rajveer18/multilevel-queue-scheduling-algorithm>***

**Pseudo code:-**

Define a struct named Process with attributes: priority, burst\_time, turnaround\_time, remaining\_time

Define a struct named Queue with attributes: priority\_start, priority\_end, total\_time, length, processes, executed

Function isNotComplete(Queue q[])

Set notComplete to false

For each queue in q:

Set countInc to 0

For each process in queue:

If process.remaining\_time is not 0:

Set notComplete to true

Else:

Increment countInc by 1

If countInc is equal to the length of the queue:

Set queue.executed to true

Return notComplete

Function sortProcessesByPriority(Queue q)

For i from 1 to q.length - 1:

For j from 0 to q.length - 1:

If q.processes[j].priority < q.processes[j + 1].priority:

Swap q.processes[j] with q.processes[j + 1]

Function updateTotalTime(Queue q[])

For each queue in q:

If queue.executed is false:

For each process in queue:

If process.remaining\_time is not 0:

Increment process.turnaround\_time by 1

Increment queue.total\_time by 1

Main function:

Define an array of Queue named queues with 3 elements

Initialize queues[0] with priority\_start 7, priority\_end 9

Initialize queues[1] with priority\_start 4, priority\_end 6

Initialize queues[2] with priority\_start 1, priority\_end 3

Prompt the user to enter the number of processes

Read and store the number of processes as num\_processes

Define an array of Process named processes with num\_processes elements

For each process in processes:

Prompt the user to enter the priority of the process

Prompt the user to enter the burst time of the process

Store the priority and burst\_time in the process

Set turnaround\_time of the process to burst\_time

For each queue in queues:

If process.priority is within the priority range of the queue:

Increment the length of the queue

For each queue in queues:

Allocate memory for processes in the queue based on the length

Initialize indices q1\_idx, q2\_idx, and q3\_idx to 0

For each process in processes:

If process.priority is within the range of the first queue:

Store the process in the first queue and increment q1\_idx

Else if process.priority is within the range of the second queue:

Store the process in the second queue and increment q2\_idx

Else:

Store the process in the third queue and increment q3\_idx

Adjust the indices by decrementing them

For each queue in queues:

Print the initial state of the queue

Initialize timer to 0, current\_queue to -1, rr\_timer to 4, counter to 0, counter\_ps to 0, and counter\_fcfs to 0

While isNotComplete(queues) is true:

If timer is equal to 10:

Set timer to 0

Increment current\_queue by 1

If current\_queue is greater than or equal to 3:

Set current\_queue to current\_queue modulo 3

If the current queue is executed:

Print that the queue is completed

Increment current\_queue by 1

If current\_queue is greater than or equal to 3:

Set current\_queue to current\_queue modulo 3

Continue

If the current queue is the first queue:

Print that the first queue is in focus

If rr\_timer is equal to 0:

Set rr\_timer to 4

For each process in the first queue:

If process.remaining\_time is 0:

Increment counter by 1

Continue

If counter is equal to the length of the first queue:

Break

While rr\_timer is greater than 0 and process.remaining\_time is not 0 and timer is not equal to 10:

Print that the first queue is executing the process

Decrement process.remaining\_time

Call updateTotalTime function

Decrement rr\_timer

Increment timer

If timer is equal to 10:

Break

If process.remaining\_time is 0 and rr\_timer is 0:

Set rr\_timer to 4

If the current process is the last in the first queue:

Set i to -1

Continue

If process.remaining\_time is 0 and rr\_timer is greater than 0:

If the current process is the last in the first queue:

Set i to -1

Continue

If rr\_timer is less than or equal to 0:

Set rr\_timer to 4

If the current process is the last in the first queue:

Set i to -1

Continue

Else if the current queue is the second queue:

Print that the second queue is in focus

Call sortProcessesByPriority for the second queue

For each process in the second queue:

If process.remaining\_time is 0:

Increment counter\_ps by 1

Continue

If counter\_ps is equal to the length of the second queue:

Break

While process.remaining\_time is not 0 and timer is not equal to 10:

Print that the second queue is executing the process

**OUTPUT**



