CSE316
SIMULATION BASED ASSIGNMENT
CA-3

NAME: Chetan Bedi
REG NO.: 12103994
SECTION: k21xr
ROLL NO.: RK21XRB24

GitHub link - https://github.com/CHETANBEDI/os.git

Q6. 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.

## The methodology adopted to solve this problem

1. Define the three queues and the priority ranges associated with each queue.
2. Prompt the user to enter the number of processes and their details such as the priority and burst time, and then assign each process to the appropriate queue based on its priority range.
3. Set the quantum time for each queue.
4. Implement the Round Robin algorithm on the highest priority queue until all processes in that queue have completed execution.
5. Implement the priority scheduling algorithm on the medium priority queue until all processes in that queue have completed execution.
6. Implement the First Come First Serve algorithm on the lowest priority queue until all processes in that queue have completed execution.
7. Repeat steps 4-6 until all processes have completed execution.

## CODE

```
import java.util.*;

public class MultilevelQueueScheduling {
    public static void main(String[] args) {

        // Define the three queues
        List<int[]> highPriorityQueue = new ArrayList<>();
        List<int[]> mediumPriorityQueue = new ArrayList<>();
```

```
        List<int[]> lowPriorityQueue = new ArrayList<>();

        // Define the priority ranges for each queue
        final int[] HIGH_PRIORITY_RANGE = {1, 3};
        final int[] MEDIUM_PRIORITY_RANGE = {4, 6};
        final int[] LOW_PRIORITY_RANGE = {7, 9};

        // Prompt the user to enter the number of processes and their details
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of processes: ");
        int numProcesses = sc.nextInt();
        for (int i = 0; i < numProcesses; i++) {
            System.out.print("Enter the priority for process " + (i + 1) + ":
");
            int priority = sc.nextInt();
            System.out.print("Enter the burst time for process " + (i + 1) + ":
");
            int burstTime = sc.nextInt();

            // Assign the process to the appropriate queue based on its priority
range
            if (priority >= HIGH_PRIORITY_RANGE[0] && priority <=
HIGH_PRIORITY_RANGE[1]) {
                highPriorityQueue.add(new int[]{i + 1, priority, burstTime});
            } else if (priority >= MEDIUM_PRIORITY_RANGE[0] && priority <=
MEDIUM_PRIORITY_RANGE[1]) {
                mediumPriorityQueue.add(new int[]{i + 1, priority, burstTime});
            } else if (priority >= LOW_PRIORITY_RANGE[0] && priority <=
LOW_PRIORITY_RANGE[1]) {
                lowPriorityQueue.add(new int[]{i + 1, priority, burstTime});
            }
        }

        // Set the quantum time for each queue
        int highPriorityQuantum = 4;
        int mediumPriorityQuantum = 10;
        int lowPriorityQuantum = 10;

        // Implement the Round Robin algorithm on the high priority queue
        while (!highPriorityQueue.isEmpty()) {
            int[] process = highPriorityQueue.remove(0);
            int processIndex = process[0];
            int priority = process[1];
            int burstTime = process[2];
            for (int i = 0; i < highPriorityQuantum; i++) {
                if (burstTime > 0) {
                    burstTime--;
                    // Check if the process is completed
                    if (burstTime == 0) {
                        // Process completed, print the details
                        System.out.println("Process " + processIndex + "
(Priority: " + priority + ") completed");
                        break;
                    }
                } else {
                    // Process completed, print the details
                    System.out.println("Process " + processIndex + " (Priority:
" + priority + ") completed");
                    break;
                }
            }

            // Check if the process still needs CPU time
            if (burstTime > 0) {
```

```
                highPriorityQueue.add(new int[]{processIndex, priority,
burstTime});
            }
        }

        // Implement the priority scheduling algorithm on the medium priority
queue
        while (!mediumPriorityQueue.isEmpty()) {
            // Find the process with the highest priority
            int[] highestPriorityProcess = mediumPriorityQueue.get(0);
            for (int i = 1; i < mediumPriorityQueue.size(); i++) {
                int[] process = mediumPriorityQueue.get(i);
                if (process[1] > highestPriorityProcess[1]) {
                    highestPriorityProcess = process;
                }

            }
        }
    }}
```

## OUTPUT SNAPSHOT