OPEN ENDED EXPERIMENT

ROUND ROBIN DISPATCHER

AIM: To design and develop Round Robin Dispatcher

SOFTWARE USED: Visual Studio Code

PROGRAM:

```
import java.util.*;

class Process {
    private int pid;
    private int priority;
    private int arrivalTime;
    private int burstTime;
    private int waitTime;
    private int exitTime;

    public Process(int pid, int priority, int arrivalTime, int burstTime) {
        this.pid = pid;
        this.priority = priority;
        this.arrivalTime = arrivalTime;
        this.burstTime = burstTime;
        this.waitTime = 0;
        this.exitTime = 0;
    }

    public int getPid() {
        return pid;
    }

    public int getPriority() {
        return priority;
    }

    public int getArrivalTime() {
        return arrivalTime;
    }

    public int getBurstTime() {
        return burstTime;
    }
```

```
    public int getWaitTime() {
        return waitTime;
    }

    public void setWaitTime(int waitTime) {
        this.waitTime = waitTime;
    }

    public int getExitTime() {
        return exitTime;
    }

    public void setExitTime(int exitTime) {
        this.exitTime = exitTime;
    }
}

class RoundRobinDispatcher {
    private Queue<Process> processQueue;
    private int timeQuantum;

    public RoundRobinDispatcher(int timeQuantum) {
        this.timeQuantum = timeQuantum;
        processQueue = new LinkedList<>();
    }

    public void addProcess(int pid, int priority, int arrivalTime, int
burstTime) {
        processQueue.add(new Process(pid, priority, arrivalTime, burstTime));
    }

    public void run() {
        int time = 0;
        int numProcesses = processQueue.size();
        int[] remainingTime = new int[numProcesses];
        int[] completionTime = new int[numProcesses];
        int[] waitTime = new int[numProcesses];
        int[] turnAroundTime = new int[numProcesses];

        for (int i = 0; i < numProcesses; i++) {
            remainingTime[i] = processQueue.peek().getBurstTime();
        }

        while (!processQueue.isEmpty()) {
            Process currentProcess = processQueue.poll();
            int index = currentProcess.getPid() - 1;
            if (remainingTime[index] <= timeQuantum) {
```

```
                time += remainingTime[index];
                completionTime[index] = time;
                turnAroundTime[index] = completionTime[index] -
currentProcess.getArrivalTime();
                waitTime[index] = turnAroundTime[index] -
currentProcess.getBurstTime();
                System.out.printf("%-12d %-12d %-12d %-12d %-12d %-12d\n",
currentProcess.getPid(), timeQuantum,
                        currentProcess.getArrivalTime(),
currentProcess.getBurstTime(), waitTime[index],
                        completionTime[index]);
            } else {
                time += timeQuantum;
                remainingTime[index] -= timeQuantum;
                processQueue.add(currentProcess);
            }
        }
    }
}

public class rrDispatch {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the time quantum: ");
        int timeQuantum = scanner.nextInt();
        RoundRobinDispatcher dispatcher = new
RoundRobinDispatcher(timeQuantum);
        System.out.print("Enter the number of processes: ");
        int numProcesses = scanner.nextInt();
        for (int i = 1; i <= numProcesses; i++) {
            System.out.printf("Enter the arrival time and burst time for
process %d: ", i);
            int arrivalTime = scanner.nextInt();
            int burstTime = scanner.nextInt();
            dispatcher.addProcess(i, 1, arrivalTime, burstTime);
        }
        System.out.println("\nProcess ID   Time Quantum Arrival Time Burst
Time  Wait Time   Exit Time");
        System.out.println("----------------------------------------------
----------------------");
        dispatcher.run();
        scanner.close();
    }

}
```

OUTPUT:

```
Enter the time quantum: 2
Enter the number of processes: 3
Enter the arrival time and burst time for process 1: 0 8
Enter the arrival time and burst time for process 2: 1 4
Enter the arrival time and burst time for process 3: 2 2

Process ID   Time Quantum Arrival Time Burst Time  Wait Time   Exit Time
-------------------------------------------------------------------------
1            2            0            8           12          20
2            2            1            4           17          22
3            2            2            2           20          24
```

CONCLUSION:

The Round Robin algorithm has been successfully implemented and executed for the given input. The output shows the scheduling of processes with their arrival time, burst time, wait time, exit time, and the time for which each process has run in milliseconds. The output indicates that the Round Robin algorithm ensures that no process is left waiting for a long time, as each process is allocated a small amount of CPU time, and the scheduler switches between the processes in a cyclic manner.