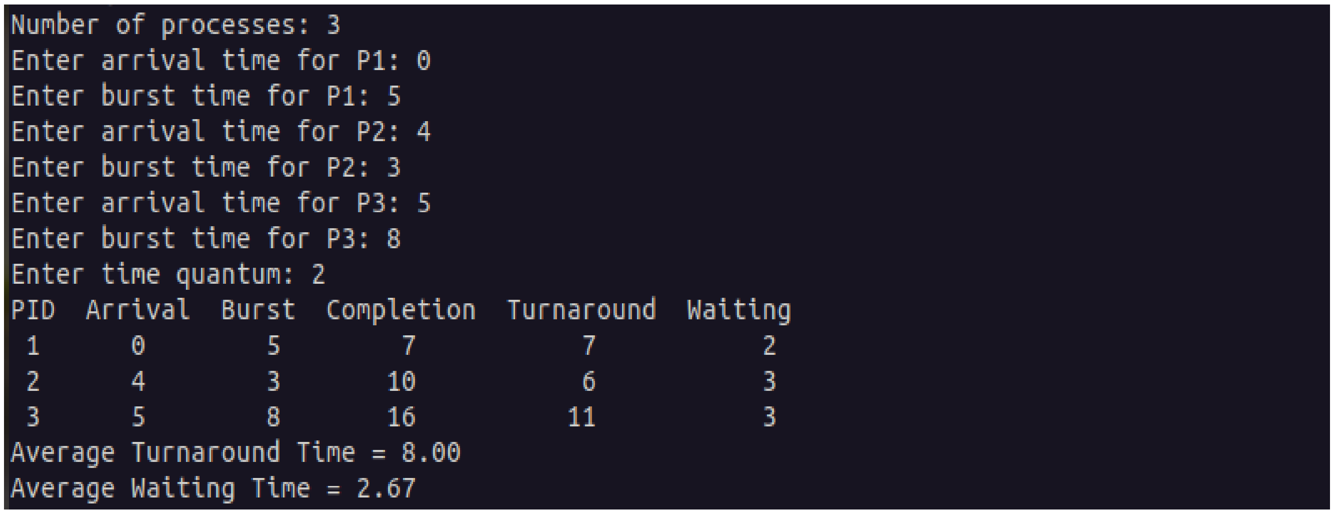
NAME:- Shaikh Areeba Mohammed Ismail
ROLL NO:- 46
CLASS/BATCH:- TE-B-2

Practical No:- 04
Write a program to simulate CPU Scheduling Algorithms: FCFS, SJF (Preemptive), Priority (Non-Preemptive) and Round Robin (Preemptive).

ROUND ROBIN (Preemptive) CODE

```
import java.util.*;
class Process { int pid; int arrivalTime; int burstTime; int
remainingTime; int completionTime; int turnaroundTime; int waitingTime;
Process(int pid, int arrivalTime, int burstTime) { this.pid = pid;
this.arrivalTime = arrivalTime; this.burstTime = burstTime;
this.remainingTime = burstTime;
}
}
public class RoundR { public static void roundRobin(List<Process>
procs, int quantum) {
// sort by arrival time procs.sort(Comparator.comparingInt(p ->
p.arrivalTime));
Queue<Process> queue = new LinkedList<>(); int currentTime = 0;
int idx = 0;  // index for next arrived process
while (true) {
// enqueue all processes that have arrived by currentTime while (idx <
procs.size() && procs.get(idx).arrivalTime <=
currentTime) { queue.add(procs.get(idx)); idx++;
}
if (queue.isEmpty()) {
// if no ready process, advance currentTime to next
arrival (if exists) if (idx < procs.size()) { currentTime =
procs.get(idx).arrivalTime; continue;
} else { break;  // all done
}
}
Process p = queue.poll();
int exec = Math.min(quantum, p.remainingTime);
p.remainingTime -= exec; currentTime += exec;
// enqueue any process that arrived during this execution
interval
while (idx < procs.size() && procs.get(idx).arrivalTime <=
currentTime) { queue.add(procs.get(idx)); idx++;
}
if (p.remainingTime > 0) {
// not finished, go back into queue queue.add(p); } else { // finished
p.completionTime = currentTime;
p.turnaroundTime = p.completionTime - p.arrivalTime;
p.waitingTime = p.turnaroundTime - p.burstTime; }
}
// output
System.out.println("PID  Arrival  Burst  Completion  Turnaround
Waiting"); double totalTAT = 0, totalWT = 0; for (Process p : procs) {
System.out.printf("%2d    %3d      %3d      %3d         %3d
%3d\n",
p.pid, p.arrivalTime, p.burstTime, p.completionTime,
p.turnaroundTime, p.waitingTime);
totalTAT += p.turnaroundTime; totalWT += p.waitingTime;
}
int n = procs.size();
System.out.printf("Average Turnaround Time = %.2f\n", totalTAT / n);
System.out.printf("Average Waiting Time = %.2f\n", totalWT / n);
}
public static void main(String[] args) { Scanner sc = new
Scanner(System.in); System.out.print("Number of processes: "); int n =
sc.nextInt();
List<Process> procs = new ArrayList<>(); for (int i = 0; i < n; i++) {
System.out.printf("Enter arrival time for P%d: ", i + 1); int at =
sc.nextInt();
System.out.printf("Enter burst time for P%d: ", i + 1); int bt =
sc.nextInt();
procs.add(new Process(i + 1, at, bt));
}
System.out.print("Enter time quantum: "); int quantum = sc.nextInt();
roundRobin(procs, quantum);
}
}
```

OUTPUT :



```
Number of processes: 3
Enter arrival time for P1: 0
Enter burst time for P1: 5
Enter arrival time for P2: 4
Enter burst time for P2: 3
Enter arrival time for P3: 5
Enter burst time for P3: 8
Enter time quantum: 2
PID  Arrival  Burst  Completion  Turnaround  Waiting
 1      0       5        7           7           2
 2      4       3       10           6           3
 3      5       8       16          11           3
Average Turnaround Time = 8.00
Average Waiting Time = 2.67
```