Practical No. : 4

Name: Hajare Vinay Arjun **Branch:** Computer Engineering

Div: CS-A **Roll No.:** 82 **P.R.N. No.:** 12320035

Course: Operating System

Aim: Write a C / C++/ Java program to find the average turnaround time and average waiting time for these processes with the following algorithms

1. Preemptive & non preemptive priority algorithm

2. Preemptive & non preemptive SJF algorithm

3. Round Robin algorithm with TQ =2

4. FCFS

Theory:

Runner.java –

```
package scheduling;

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

import java.util.Scanner;

public class Runner {

        public static void main(String[] args) {

                Scanner scanner = new Scanner(System.in);

                List<Process> processes = new ArrayList<>();

                System.out.println("Enter the toatal number of processes : ");

                int numProcesses = scanner.nextInt();


                for(int i = 0; i<numProcesses; i++) {

                        System.out.println("Enter the arrival time of process "+(i+1)+" : ");

                        int arrivalTime = scanner.nextInt();

                        System.out.println("Enter the brust time of Process "+(i+1)+" : ");
```

```
            int brustTime = scanner.nextInt();
            System.out.println("Enter the priority of Process "+(i+1)+" : ");
            int priority = scanner.nextInt();
            processes.add(new Process((i+1), arrivalTime, brustTime, priority));
        }

        List<Process> processes2 = new ArrayList<Process>(processes);
        List<Process> processes3 = new ArrayList<Process>(processes);
        List<Process> processes4 = new ArrayList<Process>(processes);
        List<Process> processes5 = new ArrayList<Process>(processes);
        List<Process> processes6 = new ArrayList<Process>(processes);

        System.out.println("----First Come First Serve : ----");
        List<Process> completedProcesses = FCFS.schedule(processes);
        Process.printStatistics(completedProcesses);

        System.out.println("----Shortest Job First (Non-Preemptive) : ----");
        List<Process> completedProcesses2 = SJFNonPreemptive.schedule(processes2);
        Process.printStatistics(completedProcesses2);

        System.out.println("----Shortest Job First (Preemptive): ----");
        List<Process> completedProcesses3 = SJFPreemptive.schedule(processes3);
        Process.printStatistics(completedProcesses3);

        System.out.println("----Priority Scheduling (Non-Preemptive): ----");
        List<Process> completedProcesses4 = PriorityNonPreemptive.schedule(processes4);
        Process.printStatistics(completedProcesses4);

        System.out.println("----Priority Scheduling (Preemptive): ----");
        List<Process> completedProcesses5 = PriorityPreemptive.schedule(processes5);
        Process.printStatistics(completedProcesses5);
```

```
                System.out.println("----Round Robin : ----");
                List<Process> completedProcesses6 = RoundRobin.schedule(processes6, 2);
                Process.printStatistics(completedProcesses6);

                scanner.close();
        }

}
```

Process.java –

```
package scheduling;

import java.util.List;

public class Process {
        private int processID;
        private int arrivalTime;
        private int brustTime;
        private int remainingTime;
        private int finishTime;
        private int trunAroundTime;
        private int waitingTime;
        private int startTime;
        private int priority;


        Process(int processID, int arrivalTime, int brustTime, int priority){
                this.processID = processID;
                this.arrivalTime = arrivalTime;
                this.brustTime = brustTime;
                this.remainingTime = brustTime;
```

```
        this.priority = priority;
    }

    public int getProcessID() {
        return processID;
    }

    public void setProcessID(int processID) {
        this.processID = processID;
    }

    public int getArrivalTime() {
        return arrivalTime;
    }

    public void setArrivalTime(int arrivalTime) {
        this.arrivalTime = arrivalTime;
    }

    public int getBrustTime() {
        return brustTime;
    }

    public void setBrustTime(int brustTime) {
        this.brustTime = brustTime;
    }

    public int getRemainingTime() {
        return remainingTime;
    }
```

```
public void setRemainingTime(int remainingTime) {
	this.remainingTime = remainingTime;
}

public int getFinishTime() {
	return finishTime;
}

public void setFinishTime(int finishTime) {
	this.finishTime = finishTime;
}

public int getTrunAroundTime() {
	return trunAroundTime;
}

public void setTrunAroundTime(int trunAroundTime) {
	this.trunAroundTime = trunAroundTime;
}

public int getWaitingTime() {
	return waitingTime;
}

public void setWaitingTime(int waitingTime) {
	this.waitingTime = waitingTime;
}

public int getStartTime() {
	return startTime;
```

```
        }

        public void setStartTime(int startTime) {
                this.startTime = startTime;
        }

        public int getPriority() {
                return priority;
        }

        public void setPriority(int priority) {
                this.priority = priority;
        }

        public static void printStatistics(List<Process> completedProcesses) {
    double totalTurnaroundTime = 0;
    double totalWaitingTime = 0;

    System.out.println("\nProcess \tPriority \tArriavl Time \tBrust Time \tFinish Time \tTurnaround Time\tWaiting Time");

    for (Process p : completedProcesses) {
       int turnaroundTime = p.getFinishTime() - p.getArrivalTime();
       int waitingTime = turnaroundTime - p.getBrustTime();
       totalTurnaroundTime += turnaroundTime;
       totalWaitingTime += waitingTime;
       System.out.println(p.getProcessID() + "\t\t" + p.getPriority()+ "\t\t" + p.getArrivalTime()+ "\t\t" + p.getBrustTime()+ "\t\t" + p.getFinishTime() + "\t\t" +turnaroundTime + "\t\t" + waitingTime);
    }

    double averageTurnaroundTime = totalTurnaroundTime / completedProcesses.size();
```

```
        double averageWaitingTime = totalWaitingTime / completedProcesses.size();
        System.out.println("\nAverage Turnaround Time: " + averageTurnaroundTime);
        System.out.println("Average Waiting Time: " + averageWaitingTime);
    }

}
```

FCFS.java –

```
package scheduling;

import java.util.ArrayList;
import java.util.List;

public class FCFS {

        public static List<Process> schedule(List<Process> processes){
                int currentTime = 0;
                List<Process> completedProcesses = new ArrayList<>();

                for(Process process : processes) {
                        // If the current time is less than the arrival time of the process,
        // move the current time to the arrival time of the process
                        if(currentTime < process.getArrivalTime()) {
                                currentTime = process.getArrivalTime();
                        }
                        // Set start time of the process
                        process.setStartTime(currentTime);
                        // Update current time by adding burst time of the process
                        currentTime += process.getBrustTime();
                        // Set finish time of the process
                        process.setFinishTime(currentTime);
                        // Add the process to completed processes
```

```
                completedProcesses.add(process);
            }

            return completedProcesses;
        }
}
```

SJFNonPreemptive.java –

```
package scheduling;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;

public class SJFNonPreemptive {

    public static List<Process> schedule(List<Process> processess){
        List<Process> readyQueue = new ArrayList<>();
        List<Process> completedProcesses = new ArrayList<>();
        int currentTime = 0;

        while(!processess.isEmpty() || !readyQueue.isEmpty()) {
            while(!processess.isEmpty() && processess.get(0).getArrivalTime() <=
currentTime) {
                readyQueue.add(processess.remove(0));
            }

            readyQueue.sort(Comparator.comparingInt(p -> p.getRemainingTime()));

            if(!readyQueue.isEmpty()) {
                Process currentProcess = readyQueue.remove(0);
                currentProcess.setStartTime(currentTime);
                currentTime += currentProcess.getRemainingTime();
```

```
                    currentProcess.setFinishTime(currentTime);
                    completedProcesses.add(currentProcess);
                }else {
                    currentTime++;
                }
            }

            return completedProcesses;
        }

}
```

SJFPreemptive.java –

```
package scheduling;

import java.util.List;
import java.util.ArrayList;
import java.util.Comparator;

public class SJFPreemptive {

        public static List<Process> schedule(List<Process> processes) {
    int currentTime = 0;
    List<Process> readyQueue = new ArrayList<>();
    List<Process> completedProcesses = new ArrayList<>();

    while (!processes.isEmpty() || !readyQueue.isEmpty()) {
       if (!processes.isEmpty() && processes.get(0).getArrivalTime() <= currentTime) {
          readyQueue.add(processes.remove(0));
          readyQueue.sort(Comparator.comparingInt(p -> p.getRemainingTime()));
       }
```

```
            if (!readyQueue.isEmpty()) {
                Process currentProcess = readyQueue.get(0);
                readyQueue.remove(0);

                currentProcess.setStartTime(currentTime);

                currentTime++;

                currentProcess.setRemainingTime(currentProcess.getRemainingTime() - 1);

                if (currentProcess.getRemainingTime() == 0) {
                    currentProcess.setFinishTime(currentTime);
                    completedProcesses.add(currentProcess);
                } else {
                    readyQueue.add(currentProcess);
                    readyQueue.sort(Comparator.comparingInt(p -> p.getRemainingTime()));
                }
            } else {
                currentTime++;
            }
        }

        return completedProcesses;
    }

}
```

PriorityNonPreemptive.java –

```
package scheduling;

import java.util.ArrayList;
import java.util.Comparator;
```

```
import java.util.List;

public class PriorityNonPreemptive {
	public static List<Process> schedule(List<Process> processes){
		List<Process> completedProcesses = new ArrayList<>();
		int currentTime = 0;

		while(!processes.isEmpty()) {
			List<Process> arrivedProcesses = new ArrayList<>();
			for(Process process : processes) {
				if(process.getArrivalTime() <= currentTime) {
					arrivedProcesses.add(process);
				}
			}

			if(!arrivedProcesses.isEmpty()) {
				arrivedProcesses.sort(Comparator.comparingInt(p ->
p.getPriority()));

				Process currentProcess = arrivedProcesses.get(0);
				processes.remove(currentProcess);

				currentProcess.setStartTime(currentTime);
				currentTime += currentProcess.getRemainingTime();

				currentProcess.setFinishTime(currentTime);
				completedProcesses.add(currentProcess);
			}else {
				currentTime++;
			}
		}
		return completedProcesses;
```

```
        }
}
```

PriorityPreemptive.java –

```
package scheduling;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;

public class PriorityPreemptive {
        public static List<Process> schedule(List<Process> processess){
                int currentTime = 0;
                List<Process> readyQueue = new ArrayList<>();
                List<Process> completedProcessess = new ArrayList<>();

                while(!processess.isEmpty() || !readyQueue.isEmpty()) {
                        while(!processess.isEmpty() && processess.get(0).getArrivalTime() <=
currentTime) {
                                readyQueue.add(processess.remove(0));
                        }

                        readyQueue.sort(Comparator.comparingInt(p -> p.getPriority()));

                        if(!readyQueue.isEmpty()) {
                                Process currentProcess = readyQueue.remove(0);

                                currentProcess.setStartTime(currentTime);
        currentProcess.setRemainingTime(currentProcess.getRemainingTime() - 1);
                                currentTime++;

                                if(currentProcess.getRemainingTime() == 0){
```

```
                    currentProcess.setFinishTime(currentTime);
                    completedProcessess.add(currentProcess);
                }else {
                    readyQueue.add(currentProcess);
                }
            }else {
                currentTime++;
            }
        }
        return completedProcessess;
    }
}
```

RoundRobin.java –

```
package scheduling;

import java.util.ArrayList;
import java.util.List;

public class RoundRobin {

   public static List<Process> schedule(List<Process> processes, int quantum) {

      List<Process> readyQueue = new ArrayList<>();
      List<Process> completedProcesses = new ArrayList<>();
      int currentTime = 0;

      while (!processes.isEmpty() || !readyQueue.isEmpty()) {
         while (!processes.isEmpty() && processes.get(0).getArrivalTime() <= currentTime) {
            readyQueue.add(processes.remove(0));
         }
```

```
            if (!readyQueue.isEmpty()) {
                Process currentProcess = readyQueue.remove(0);
                currentProcess.setStartTime(currentTime);
                int remainingBurstTime = currentProcess.getRemainingTime();

                if (remainingBurstTime <= quantum) {
                    // Process completes within the time quantum
                    currentTime += remainingBurstTime;
                    currentProcess.setFinishTime(currentTime);
                    currentProcess.setRemainingTime(0);
                    completedProcesses.add(currentProcess);
                } else {
                    // Process needs more quantum time, but we have to yield to the next process after
quantum
                    currentTime += quantum;
                    currentProcess.setRemainingTime(currentProcess.getRemainingTime() - quantum);
                    readyQueue.add(currentProcess); // Adding back to the end of the queue
                }
            } else {
                currentTime++;
            }
        }

        return completedProcesses;
    }
}
```

Output :

```
Enter the toatal number of processes :
5
Enter the arrival time of process 1 :
0
Enter the brust time of Process 1 :
4
Enter the priority of Process 1 :
1
Enter the arrival time of process 2 :
0
Enter the brust time of Process 2 :
3
Enter the priority of Process 2 :
2
Enter the arrival time of process 3 :
6
Enter the brust time of Process 3 :
7
Enter the priority of Process 3 :
1
Enter the arrival time of process 4 :
11
Enter the brust time of Process 4 :
4
Enter the priority of Process 4 :
3
Enter the arrival time of process 5 :
12
Enter the brust time of Process 5 :
2
Enter the priority of Process 5 :
2
----First Come First Serve : ----

Process  Priority     Arriavl Time     Brust Time
     Finish Time  Turnaround Time  Waiting Time
1        1        0        4        4        4        0
2        2        0        3        7        7        4
3        1        6        7        14       8        1
4        3        11       4        18       7        3
5        2        12       2        20       8        6
```

```
Average Turnaround Time: 6.8
Average Waiting Time: 2.8

----Shortest Job First (Non-Preemptive) : ----

Process  Priority     Arriavl Time     Brust Time
    Finish Time  Turnaround Time  Waiting Time
2        2        0        3        3        3        0
1        1        0        4        7        7        3
3        1        6        7        14       8        1
5        2        12       2        16       4        2
4        3        11       4        20       9        5

Average Turnaround Time: 6.2
Average Waiting Time: 2.2

----Shortest Job First (Preemptive): ----

Process  Priority     Arriavl Time     Brust Time
    Finish Time  Turnaround Time  Waiting Time
1        1        0        4        4        4        0
2        2        0        3        7        7        4
3        1        6        7        14       8        1
5        2        12       2        16       4        2
4        3        11       4        20       9        5

Average Turnaround Time: 6.4
Average Waiting Time: 2.4

----Priority Scheduling (Non-Preemptive): ----

Process  Priority     Arriavl Time     Brust Time
    Finish Time  Turnaround Time  Waiting Time
1        1        0        4        4        4        0
2        2        0        3        7        7        4
3        1        6        7        14       8        1
5        2        12       2        16       4        2
4        3        11       4        20       9        5

Average Turnaround Time: 6.4
Average Waiting Time: 2.4

----Priority Scheduling (Preemptive): ----
```

```
Process  Priority     Arriavl Time     Brust Time
    Finish Time  Turnaround Time  Waiting Time
1        1        0        4        4        4        0
3        1        6        7        13       7        0
2        2        0        3        14       14       11
5        2        12       2        16       4        2
4        3        11       4        20       9        5

Average Turnaround Time: 7.6
Average Waiting Time: 3.6

----Round Robin : ----

Process  Priority     Arriavl Time     Brust Time
    Finish Time  Turnaround Time  Waiting Time
1        1        0        4        6        6        2
2        2        0        3        7        7        4
3        1        6        7        16       10       3
5        2        12       2        18       6        4
4        3        11       4        20       9        5

Average Turnaround Time: 7.6
Average Waiting Time: 3.6
```