Experiment 2.1

Student Name: Saurav Suryavanshi **UID:**21BCS3460
Branch: BE-CSE **Section/Group:**NTPP_601_A
Date of Performance: 27-02-2024 **Semester:** 6th
Subject Name: Cloud Computing & Distributed Systems Lab
Subject Code: 21CSP-378

1. **Aim:** Simulate a cloud scenario using Matlab and run a scheduling algorithm.

2. **Objective:** Develop a MATLAB simulation for cloud computing, implementing and evaluating diverse scheduling algorithms. Assess performance metrics such as task completion time and resource utilization. Analyze results to understand algorithm efficiency, optimize strategies, and contribute insights for effective cloud resource allocation.

3. **Theory:** Cloud computing simulation in MATLAB involves modeling VMs, tasks, and network latency. Various scheduling algorithms like Round Robin are implemented, assessing metrics such as task completion time and resource utilization. The simulation aims to optimize algorithms, providing insights for efficient cloud resource allocation in diverse workloads.

4. Procedure:

Step 1: Write down the java code for executing FCFS scheduling algorithms

```
import java.text.ParseException;

class GFG {

    // Function to find the waiting time for all
    // processes
    static void findWaitingTime(int processes[], int n,
            int bt[], int wt[]) {
        // waiting time for first process is 0wt[0]
        = 0;

        // calculating waiting time
        for (int i = 1; i < n; i++)
            { wt[i] = bt[i - 1] + wt[i -
            1];
```

```
        }
    }

    // Function to calculate turn around time
    static void findTurnAroundTime(int processes[], int n,
            int bt[], int wt[], int tat[]) {
        // calculating turnaround time by adding
        // bt[i] + wt[i]
        for (int i = 0; i < n; i++)
            {tat[i] = bt[i] + wt[i];
        }
    }

    //Function to calculate average time
    static void findavgTime(int processes[], int n, int bt[]) {
        int wt[] = new int[n], tat[] = new int[n];
        int total_wt = 0, total_tat = 0;
        //Function to find waiting time of all processes
        findWaitingTime(processes, n, bt, wt);
        //Function to find turn around time for all processes
        findTurnAroundTime(processes, n, bt, wt, tat);
        //Display processes along with all details
        System.out.printf("Processes  Burst  time  Waiting"
                          +" time Turn around time\n");
        // Calculate total waiting time and total turn
        // around time
        for (int i = 0; i < n; i++)
            { total_wt = total_wt + wt[i];
            total_tat = total_tat + tat[i];
            System.out.printf(" %d ", (i + 1));
            System.out.printf("     %d ", bt[i]);
            System.out.printf("     %d", wt[i]);
            System.out.printf("     %d\n", tat[i]);
        }
        float s = (float)total_wt /(float) n;
        int t = total_tat / n;
        System.out.printf("Average waiting time = %f", s);
        System.out.printf("\n");
        System.out.printf("Average turn around time = %d ", t);
    }
     // Driver code
    public static void main(String[] args) throws ParseException {
        //process id's
        int processes[] = {1, 2, 3};
        int n = processes.length;
          //Burst time of all processes
         int burst_time[] = {10, 5, 8};

        findavgTime(processes,  n,  burst_time);

    }
}
```

Step 2: Write down the java code for executing SJF scheduling algorithms

```
import java.io.*;
import java.util.*;

class Main {
    public static void main(String[] args)
    {
        Scanner input = new Scanner(System.in);
        int n;
        // Matrix for storing Process Id, Burst
        // Time, Average Waiting Time & Average
        // Turn Around Time.
        int[][] A = new int[100][4];
        int total = 0;
        float avg_wt, avg_tat;
        System.out.println("Enter number of process:");
        n = input.nextInt();
        System.out.println("Enter Burst Time:");
        for (int i = 0; i < n; i++) {
            // User Input Burst Time and alloting
            // Process Id.
            System.out.print("P" + (i + 1) + ": ");A[i][1]
            = input.nextInt();
            A[i][0] = i + 1;
        }
        for (int i = 0; i < n; i++) {
            // Sorting process according to their
            // Burst Time.
            int index = i;
            for (int j = i + 1; j < n; j++) {
                if (A[j][1] < A[index][1])
                    {index = j;
                }
            }
            int temp = A[i][1];
            A[i][1] = A[index][1];
            A[index][1] = temp;
            temp = A[i][0];
            A[i][0] = A[index][0];
            A[index][0] = temp;
        }
        A[0][2] = 0;
        // Calculation of Waiting Times
        for (int i = 1; i < n; i++)
            {A[i][2] = 0;
            for (int j = 0; j < i; j++)
                {A[i][2] += A[j][1];
            }
            total += A[i][2];
        }
        avg_wt = (float)total / n;
        total = 0;
        // Calculation of Turn Around Time and printing the
        // data.
```

```
        System.out.println("P\tBT\tWT\tTAT");
        for (int i = 0; i < n; i++)
            { A[i][3] = A[i][1] +
            A[i][2];
            total += A[i][3];
            System.out.println("P" + A[i][0] + "\t"
                               + A[i][1] + "\t" + A[i][2]
                               + "\t" + A[i][3]);
        }
        avg_tat = (float)total / n;
        System.out.println("Average Waiting Time= "
                           + avg_wt);
        System.out.println("Average Turnaround Time= "
                           + avg_tat);
    }
}
```

Step 3: Write down the java code for executing Round Robin scheduling algorithms

```
public class GFG
{
    // Method to find the waiting time for all
    // processes
    static void findWaitingTime(int processes[], int n,
                  int bt[], int wt[], int quantum)
    {
        // Make a copy of burst times bt[] to store remaining
        // burst times.
        int rem_bt[] = new int[n];
        for (int i = 0 ; i < n ; i++)
            rem_bt[i] =  bt[i];

        int t = 0; // Current time

        // Keep traversing processes in round robin manner
        // until all of them are not done.
        while(true)
        {
            boolean done = true;

            // Traverse all processes one by one repeatedly
            for (int i = 0 ; i < n; i++)
            {
                // If burst time of a process is greater than 0
                // then only need to process further
                if (rem_bt[i] > 0)
                {
                    done = false; // There is a pending process

                    if (rem_bt[i] > quantum)
                    {
                        // Increase the value of t i.e. shows
                        // how much time a process has been processed
                        t += quantum;
```

```
                    // Decrease the burst_time of current process
                    // by quantum
                    rem_bt[i] -= quantum;
                }

                // If burst time is smaller than or equal to
                // quantum. Last cycle for this process
                else
                {
                    // Increase the value of t i.e. shows
                    // how much time a process has been processed
                    t = t + rem_bt[i];

                    // Waiting time is current time minus time
                    // used by this process
                    wt[i] = t - bt[i];

                    // As the process gets fully executed
                    // make its remaining burst time = 0
                    rem_bt[i] = 0;
                }
            }
        }

        // If all processes are done
        if (done == true)
          break;
    }
}

// Method to calculate turn around time
static void findTurnAroundTime(int processes[], int n,
                        int bt[], int wt[], int tat[])
{
    // calculating turnaround time by adding
    // bt[i] + wt[i]
    for (int i = 0; i < n ; i++)
        tat[i] = bt[i] + wt[i];
}

// Method to calculate average time
static void findavgTime(int processes[], int n, int bt[],
                                        int quantum)
{
    int wt[] = new int[n], tat[] = new int[n];
    int total_wt = 0, total_tat = 0;

    // Function to find waiting time of all processes
    findWaitingTime(processes, n, bt, wt, quantum);

    // Function to find turn around time for all processes
    findTurnAroundTime(processes, n, bt, wt, tat);
```

```
        // Display processes along with all details
        System.out.println("PN " + " B " +
                           " WT " + " TAT");

        // Calculate total waiting time and total turn
        // around time
        for (int i=0; i<n; i++)
        {
            total_wt = total_wt + wt[i];
            total_tat = total_tat + tat[i];
            System.out.println(" " + (i+1) + "\t\t" + bt[i] +"\t " +
                               wt[i] +"\t\t " + tat[i]);
        }

        System.out.println("Average waiting time = " +
                           (float)total_wt / (float)n);
        System.out.println("Average turn around time = " +
                            (float)total_tat / (float)n);
    }

    // Driver Method
    public static void main(String[] args)
    {
        // process id's
        int processes[] = { 1, 2, 3};
        int n = processes.length;

        // Burst time of all processes
        int burst_time[] = {10, 5, 8};

        // Time quantum
        int quantum = 2;
        findavgTime(processes, n, burst_time, quantum);
    }
}
```

5. Output:-

FCFS:

```
Processes Burst time Waiting time Turn around time
 1        10         0            10
 2        5          10           15
 3        8          15           23
Average waiting time = 8.333333
Average turn around time = 16
```

SJF:

```
Enter number of process:
5
Enter Burst Time:
P1: 2
P2: 2
P3: 2
P4: 2
P5: 3
P        BT       WT       TAT
P1       2        0        2
P2       2        2        4
P3       2        4        6
P4       2        6        8
P5       3        8        11
Average Waiting Time= 4.0
Average Turnaround Time= 6.2
```

Round Robin:

```
Processes Burst time Waiting time Turn around time
 1       10        0       10
 2       5        10       15
 3       8        15       23
Average waiting time = 8.333333
Average turn around time = 16
```

6. Learning Outcome:

i). Learned how to install and use Eclipse IDE

ii). Learned how to install Cloud sim IDE and how to use it with eclipse.

iii). Learned how to simulate in Eclipse using cloud sim IDE.