Name: Subhendu Maji Roll number: 18ETCS002121

Laboratory 4

Title of the Laboratory Exercise: Programs for process scheduling algorithms

1. Introduction and Purpose of Experiment

   A Process Scheduler schedules different processes to CPU based on particular scheduling algorithms. There are various scheduling algorithms present in each group of operating system. By solving these problems students will be able use different scheduling algorithms as part of their implementation

2. Aim and Objectives

   Aim

   - To develop programs to implement scheduling algorithms

   Objectives

   At the end of this lab, the student will be able to

   - Distinguish different scheduling algorithms
   - Apply the logic of scheduling algorithms wherever required Create C programs to simulate scheduling algorithms

3. Experimental Procedure

   i. Analyse the problem statement ii. Design an algorithm for the given problem statement and develop a flowchart/pseudo-code iii. Implement the algorithm in C language iv. Compile the C program

   v. Test the implemented program vi. Document the Results vii. Analyse and discuss the outcomes of your experiment

4. Questions

   Write program to simulate the following process scheduling algorithms. Calculate average waiting time and average turnaround time for processes under each scheduling algorithm.

Instructions: Assume all the processes arrive at the same time. For FCFS and SJF scheduling algorithms, read the number of processes/jobs in the system and their CPU burst times. For round robin scheduling algorithm, read the number of processes in the system, their CPU burst times and the size of the time slice. For priority scheduling algorithm, read the number of processes in the system, their CPU burst times and the priorities.

a) First Come First Served
b) Shortest Job First
c) Priority
d) Round Robin

5. Calculations/Computations/Algorithms

Algorithm of First Come First Served (FCFS):

```
STEP 1: Start

STEP 2: Input the processes along with their burst time (bt).

STEP 3: Find waiting time (wt) for all processes.

STEP 4: As first process that comes need not to wait so waiting
time for process 1 will be 0 i.e. wt[0] = 0.

STEP 5: Find waiting time for all other processes i.e. for

     process i ->

       wt[i] = bt[i-1] + wt[i-1] .

STEP 6: Find turnaround time = waiting_time + burst_time

    for all processes.

STEP 7: Find average waiting time =

                total_waiting_time / no_of_processes.

STEP 8:   Similarly, find average turn_around_time =

                total_turn_around_time / no_of_processes.

STEP 9: Stop
```

Name: Subhendu Maji Roll number: 18ETCS002121

Algorithm of Shortest Job First (SJF):

STEP 1: Sort all the process according to the arrival time.

STEP 2: Then select that process which has minimum arrival time and minimum Burst time.

STEP 3: After completion of process make a pool of process which after till the completion of previous process and select that process among the pool which is having minimum Burst time.

Algorithm of Priority Scheduling:

STEP 1: First input the processes with their burst time and priority.

STEP 2: Sort the processes, burst time and priority according to the priority.

STEP 3: Now simply apply FCFS algorithm.

Algorithm of Shortest Job First (SJF):

STEP 1: Start

STEP 2: Create an array rem_bt[] to keep track of remaining burst time of processes. This array is initially a copy of bt[] (burst times array)

STEP 3: Create another array wt[] to store waiting times processes. Initialize this array as 0.

STEP 4: Initialize time: t = 0

STEP 5: Keep traversing the all processes while all processes are not done. Do following for i 'th process if it is not done yet.

4.1 If rem_bt[i] > quantum

4.1.1 t = t + quantum

4.1.2 bt_rem[i] -= quantum;

4.2 Else // Last cycle for this process

4.2.1 t = t + bt_rem[i];

4.2.2 wt[i] = t - bt[i]

4.2.3 bt_rem[i] = 0; // This process is over

STEP 6: Stop.

6. Presentation of Results

makefile

```
fcfs:
    @gcc fcfs.c
    @./a.out

sjf:
    @gcc sjf.c
    @./a.out

priority:
    @gcc priority.c
    @./a.out

round_robin:
    @gcc round_robin.c
    @./a.out

clean:
    @rm *.out

.PHONY: clean
```

Figure 15 makefile

```c
#include <stdio.h>

// Function to find the waiting time for all processes
void findWaitingTime(int processes[], int n, int bt[], int wt[])
{
    // waiting time for first process is 0
    wt[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
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
void findavgTime(int processes[], int n, int bt[])
{
    int wt[n], tat[n], 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
    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];
        printf("%d", (i + 1));
        printf("\t\t%d ", bt[i]);
        printf("\t\t%d", wt[i]);
        printf("\t\t%d\n", tat[i]);
    }
    int s = (float)total_wt / (float)n;
    int t = (float)total_tat / (float)n;
    printf("Average waiting time = %d", s);
    printf("\n");
    printf("Average turn around time = %d ", t);
}

// Driver code
int main()
{
    int n;
    printf("Enter number of processes: ");
    scanf("%d", &n);

    int processes[n], burst_time[n];

    //process id's
    printf("Enter Processes: ");
    for (size_t i = 0; i < n; i++)
    {
        scanf("%d", &processes[i]);
    }

    //Burst time of all processes
    printf("Enter burst times: ");
    for (size_t i = 0; i < n; i++)
    {
        scanf("%d", &burst_time[i]);
    }

    findavgTime(processes, n, burst_time);
    return 0;
}
```

Figure 16 FCFS source code

```
 subhendu@LAPTOP-AL8CTHTV  /mnt/d/RUAS-sem-05/OS/lab5  make fcfs
Enter number of processes: 4
Enter Processes: 1 2 3 4
Enter burst times: 5 1 7 8
Processes   Burst_time   Waiting_time   Turn_around_time
1               5               0               5
2               1               5               6
3               7               6               13
4               8               13              21
Average waiting time = 6
Average turn around time = 11 %
 subhendu@LAPTOP-AL8CTHTV  /mnt/d/RUAS-sem-05/OS/lab5
```

Figure 17 Execution of FCFS

```c
#include <stdio.h>
int main()
{
    int bt[20], p[20], wt[20], tat[20], i, j, n, total = 0, pos, temp;
    float avg_wt, avg_tat;
    printf("Enter number of process:");
    scanf("%d", &n);

    printf("\nEnter Burst Time:\n");
    for (i = 0; i < n; i++)
    {
        printf("p%d:", i + 1);
        scanf("%d", &bt[i]);
        p[i] = i + 1;
    }

    //sorting of burst times
    for (i = 0; i < n; i++)
    {
        pos = i;
        for (j = i + 1; j < n; j++)
        {
            if (bt[j] < bt[pos])
                pos = j;
        }

        temp = bt[i];
        bt[i] = bt[pos];
        bt[pos] = temp;

        temp = p[i];
        p[i] = p[pos];
        p[pos] = temp;
    }

    wt[0] = 0;

    for (i = 1; i < n; i++)
    {
        wt[i] = 0;
        for (j = 0; j < i; j++)
            wt[i] += bt[j];

        total += wt[i];
    }

    avg_wt = (float)total / n;
    total = 0;

    printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");
    for (i = 0; i < n; i++)
    {
        tat[i] = bt[i] + wt[i];
        total += tat[i];
        printf("\n %d \t\t  %d\t\t %d \t\t\t %d", p[i], bt[i], wt[i], tat[i]);
    }

    avg_tat = (float)total / n;
    printf("\n\nAverage Waiting Time=%0.3f", avg_wt);
    printf("\nAverage Turnaround Time=%0.3f\n", avg_tat);
}
```

Figure 18 SJF Source Code

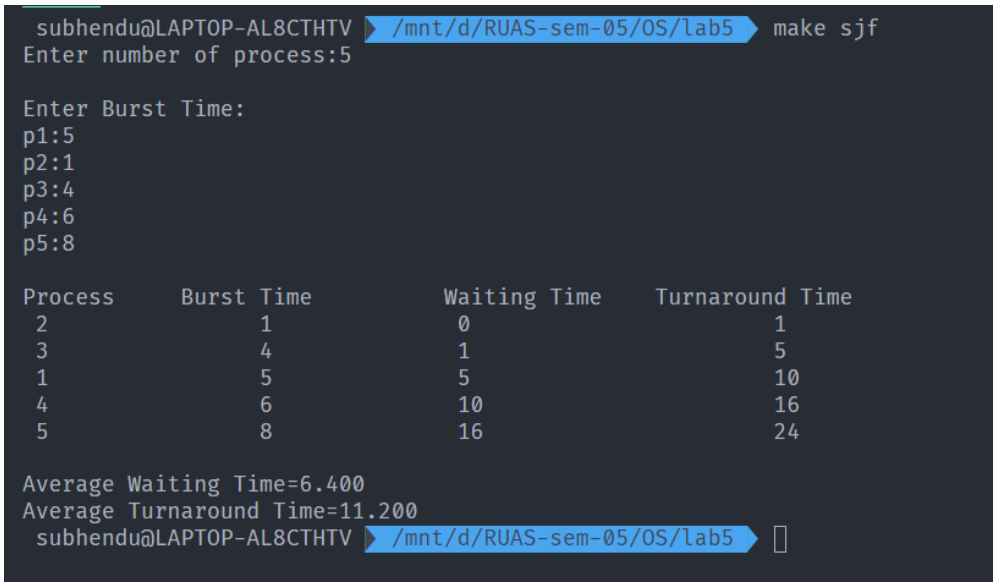

Figure 19 Execution of SJF

```c
#include <stdio.h>

int main()
{
    int bt[20], p[20], wt[20], tat[20], pr[20], i, j, n, total = 0, pos, temp;
    float avg_wt, avg_tat;
    printf("Enter Total Number of Process:");
    scanf("%d", &n);

    printf("\nEnter Burst Time and Priority\n");
    for (i = 0; i < n; i++)
    {
        printf("\nP[%d]\n", i + 1);
        printf("Burst Time:");
        scanf("%d", &bt[i]);
        printf("Priority:");
        scanf("%d", &pr[i]);
        p[i] = i + 1; //contains process number
    }

    //sorting burst time, priority and process number in ascending order using selection sort
    for (i = 0; i < n; i++)
    {
        pos = i;
        for (j = i + 1; j < n; j++)
        {
            if (pr[j] < pr[pos])
                pos = j;
        }
        // swaping
        temp = pr[i];
        pr[i] = pr[pos];
        pr[pos] = temp;

        temp = bt[i];
        bt[i] = bt[pos];
        bt[pos] = temp;

        temp = p[i];
        p[i] = p[pos];
        p[pos] = temp;
    }

    wt[0] = 0; //waiting time for first process is zero

    //calculate waiting time
    for (i = 1; i < n; i++)
    {
        wt[i] = 0;
        for (j = 0; j < i; j++)
            wt[i] += bt[j];

        total += wt[i];
    }

    avg_wt = (float)total / (float)n; //average waiting time
    total = 0;

    printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");
    for (i = 0; i < n; i++)
    {
        tat[i] = bt[i] + wt[i]; //calculate turnaround time
        total += tat[i];
        printf("\nP[%d]\t\t  %d\t\t    %d \t\t\t %d", p[i], bt[i], wt[i], tat[i]);
    }

    avg_tat = (float)total / (float)n; //average turnaround time
    printf("\n\nAverage Waiting Time=%0.3f", avg_wt);
    printf("\nAverage Turnaround Time=%0.3f\n", avg_tat);

    return 0;
}
```

Figure 20 Priority Scheduling Source Code

```
 subhendu@LAPTOP-AL8CTHTV  /mnt/d/RUAS-sem-05/OS/lab5  make priority
Enter Total Number of Process:3

Enter Burst Time and Priority

P[1]
Burst Time:6
Priority:3

P[2]
Burst Time:18
Priority:2

P[3]
Burst Time:14
Priority:1

Process     Burst Time          Waiting Time    Turnaround Time
P[3]              14                0                 14
P[2]              18                14                32
P[1]              6                 32                38

Average Waiting Time=15.333
Average Turnaround Time=28.000
 subhendu@LAPTOP-AL8CTHTV  /mnt/d/RUAS-sem-05/OS/lab5
```

Figure 21 Execution of Priority Scheduling

```c
#include <stdio.h>
#include <stdbool.h>

void getTAT(int processes[], int n, int bt[], int wt[], int tat[]);
void getWT(int processes[], int n, int bt[], int wt[], int quantum);
void getAvgTime(int processes[], int n, int bt[], int quantum);

int main()
{
    int n;
    printf("Enter no of processes:-");
    scanf("%d", &n);

    int processes[n];
    int burst_time[n];

    // Time quantum
    int quantum;
    printf("Enter Time Quantam:-");
    scanf("%d", &quantum);
    printf("Enter burst times:- \n");
    for (size_t i = 0; i < n; i++)
    {
        processes[i] = i;
        scanf(" %d", &burst_time[i]);
    }

    getAvgTime(processes, n, burst_time, quantum);
    return 0;
}

void getWT(int processes[], int n, int bt[], int wt[], int quantum)
{
    int rem_bt[n];
    for (int i = 0; i < n; i++)
        rem_bt[i] = bt[i];

    int t = 0;
    while (true)
    {
        bool done = true;
        for (int i = 0; i < n; i++)
        {
            if (rem_bt[i] > 0)
            {
                done = false;
                if (rem_bt[i] > quantum)
                {
                    t += quantum;
                    rem_bt[i] -= quantum;
                }
                else
                {
                    t = t + rem_bt[i];
                    wt[i] = t - bt[i];
                    rem_bt[i] = 0;
                }
            }
        }

        if (done == true)
            break;
    }
}

void getTAT(int processes[], int n, int bt[], int wt[], int tat[])
{
    for (int i = 0; i < n; i++)
        tat[i] = bt[i] + wt[i];
}
void getAvgTime(int processes[], int n, int bt[], int quantum)
{
    int wt[n], tat[n], total_wt = 0, total_tat = 0;

    getWT(processes, n, bt, wt, quantum);
    getTAT(processes, n, bt, wt, tat);

    printf("Prs \t BT \t WT \t TAT\n");

    // Calculate total WT and Total TAT
    for (int i = 0; i < n; i++)
    {
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        printf("%d \t %d \t %d \t %d \n", i + 1, bt[i], wt[i], tat[i]);
    }
    printf("Average waiting time = %0.3f", (float)total_wt / (float)n);
    printf("\nAverage turn around time = %0.3f", (float)total_tat / (float)n);
}
```

Figure 22 Round Robin Source Code

```
 subhendu@LAPTOP-AL8CTHTV  /mnt/d/RUAS-sem-05/OS/lab5  make round_robin
Enter no of processes:-4
Enter Time Quantam:-3
Enter burst times:-
8 12 16 20
Prs      BT      WT      TAT
1        8       18      26
2        12      26      38
3        16      35      51
4        20      36      56
Average waiting time = 28.750
Average turn around time = 42.750%
 subhendu@LAPTOP-AL8CTHTV  /mnt/d/RUAS-sem-05/OS/lab5
```

Figure 23 Execution of Round Robin

7. Analysis and Discussions

- **First Come First Served**

  Given n processes with their burst times, the task is to find average waiting time and average turnaround time using FCFS scheduling algorithm.
  First in, first out (FIFO), also known as first come, first served (FCFS), is the simplest scheduling algorithm. FIFO simply queues processes in the order that they arrive in the ready queue.
  In this, the process that comes first will be executed first and next process starts only after the previous gets fully executed.

- **Shortest Job First**

  Shortest job first (SJF) or shortest job next, is a scheduling policy that selects the waiting process with the smallest execution time to execute next. SJN is a non-pre-emptive algorithm.

  - Shortest Job first has the advantage of having a minimum average waiting time among all scheduling algorithms.
  - It is a Greedy Algorithm.
  - It may cause starvation if shorter processes keep coming. This problem can be solved using the concept of ageing.
  - It is practically infeasible as Operating System may not know burst time and therefore may not sort them. While it is not possible to predict execution time, several methods can be used to estimate the execution time for a job, such as a weighted average of previous execution times. SJF can be used in specialized environments where accurate estimates of running time are available.

- **Priority**

  Priority scheduling is one of the most common scheduling algorithms in batch systems. Each process is assigned a priority. Process with the highest priority is to be executed first and so on.

Processes with the same priority are executed on first come first served basis. Priority can be decided based on memory requirements, time requirements or any other resource requirement

- **Round Robin**

  Round Robin is a CPU scheduling algorithm where each process is assigned a fixed time slot in a cyclic way.

  - It is simple, easy to implement, and starvation-free as all processes get fair share of CPU.
  - One of the most commonly used technique in CPU scheduling as a core.
  - It is pre-emptive as processes are assigned CPU only for a fixed slice of time at most.
  - The disadvantage of it is more overhead of context switching.

8. Conclusions

In FCFS,
It is simple and easy to understand

In SJF,

- Shortest jobs are favoured.
- It is provably optimal; in that it gives the minimum average waiting time for a given set of processes.

In Round Robin,

- Every process gets an equal share of the CPU.
- RR is cyclic in nature, so there is no starvation

In Priority,

This Scheduling provides a good mechanism where the relative importance of each process maybe precisely defined.

9. Comments

1. Limitations of Experiments

FCFS limitation

In First Come First Serve scheduling the average waiting time is not optimal. ...

The First Come First Serve scheduling is Non-pre-emptive in nature. ...

As the First come First Serve scheduling is Non-pre-emptive, it does not understand the priority of processes.

SJF Limitations

SJF may cause starvation, if shorter processes keep coming. This problem is solved by aging. It cannot be implemented at the level of short-term CPU scheduling.

Round Robin limitation

If slicing time of OS is low, the processor output will be reduced. This method spends more time on context switching. Its performance heavily depends on time quantum. Priorities cannot be set for the processes

Priority scheduling limitation

The problem occurs when the operating system gives a particular task a very low priority, so it sits in the queue for a larger amount of time, not being dealt with by the CPU. If this process is something the user needs, there could be a very long wait, this process is known as “Starvation” or “Infinite Blocking”.

2. Learning happened

Implemented scheduling algorithms in C.