Experiment No. 7

Title: Implement any one non pre-emptive and one pre-emptive Scheduling algorithm

Batch: B2 **Roll No: 16010420117** **Experiment No: 7**

Aim: To implement any one non pre-emptive and pre-emptive scheduling algorithm.

Resources needed: Any Java/C editor and compiler

Theory:

Pre lab/Prior concepts:

Non-preemptive Scheduling algorithms are designed so that once a process enters the running state; it cannot be preempted until it completes its allotted time whereas the preemptive scheduling is based on priority where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state.

CPU scheduling decisions take place under one of four conditions:

1. When a process switches from running state to the waiting state such as for an I/O request.
2. When a process switches from running state to the ready state, for example in response to an interrupt.
3. When a process switches from waiting state to the ready state, say at completion of I/O.
4. When a process terminates.

If scheduling takes place only under conditions 1 and 4, the system is said to be non-preemptive, or cooperative. Under these conditions, once a process starts running it keeps running, until it either voluntarily blocks or until it finishes. Otherwise the system is said to be preemptive.
Windows used non-preemptive Scheduling up to windows 3.x, and started using preemptive scheduling with Win95.Mac used non preemptive prior to OSX, and preemptive since then. Following are some non-Preemptive Scheduling algorithms.

Turn Around Time: In computing, turnaround time is the total time taken between the submission of a program/process/thread/task (Linux) for execution and the return of the complete output to the customer/user. It may vary for various programming languages depending on the developer of the software or the program.

Waiting time: A waiting period is the period of time between when an action is requested or mandated and when it occurs.

Burst time: CPU burst. CPU burst: the amount of time the process uses the processor before it is no longer ready. Types of CPU bursts: long bursts -- process is CPU bound (i.e. array work).

FCFS Non-Preemptive:
1) Create process with PID and CPU Burst time.
2) Put in Ready queue.
3) Take one by one process for execution from the ready queue
4) Show execution of processes (Gantt Chart)
5) Calculate Average waiting time.
6) Calculate Average Turnaround time.

SJF Non-Preemptive :
1) Create process with PID and CPU Burst time.
2) Sort the processes according to CPU Burst time and put in Ready queue.
3) Take one by one process for execution from the ready queue
4) Show execution of processes (Gantt Chart)
5) Calculate Average waiting time.
6) Calculate Average Turnaround time.

Activities:

1. Students have to study different non pre-emptive, pre-emptive scheduling algorithms and implement any one of them.
2. Calculate average waiting time and average turnaround time of the algorithm.

Results: Attach the results in a separate document. (No snapshots to be attached)

This file must contain on the top:
Name:
Roll No.
Exp No.
Batch:
Date:

Students have to upload this document electronically.

Q1. Students have to study different non pre-emptive, pre-emptive scheduling algorithms and implement any one of them.
Q2.Calculate average waiting time and average turnaround time of the algorithm.
Solution:

FCFS-First Come First Serve Scheduling Algorithm- Non-Preemptive

CODE(PYTHON)

1. **FCFS-First Come First Serve Scheduling Algorithm- Non-Preemptive**

CODE(C):

```
#include <stdio.h>
#include <stdlib.h>

#define MAX 100

typedef struct
{
   int pid;
   int burst_time;
   int waiting_time;
   int turnaround_time;
} Process;

void print_table(Process p[], int n);
void print_gantt_chart(Process p[], int n);

int main()
{
   Process p[MAX];
   int i, j, n;
   int sum_waiting_time = 0, sum_turnaround_time;
   printf("Enter total number of process: ");
   scanf("%d", &n);
   printf("Enter burst time for each process:\n");
   for(i=0; i<n; i++) {
```

```
        p[i].pid = i+1;
        printf("P[%d] : ", i+1);
        scanf("%d", &p[i].burst_time);
        p[i].waiting_time = p[i].turnaround_time = 0;
    }

    // calculate waiting time and turnaround time
    p[0].turnaround_time = p[0].burst_time;

    for(i=1; i<n; i++) {
        p[i].waiting_time = p[i-1].waiting_time + p[i-1].burst_time;
        p[i].turnaround_time = p[i].waiting_time + p[i].burst_time;
    }

    // calculate sum of waiting time and sum of turnaround time
    for(i=0; i<n; i++) {
            sum_waiting_time += p[i].waiting_time;
            sum_turnaround_time += p[i].turnaround_time;
    }

    // print table
    puts(""); // Empty line
    print_table(p, n);
    puts(""); // Empty Line
    printf("Total Waiting Time      : %-2d\n", sum_waiting_time);
    printf("Average Waiting Time    : %-2.2lf\n", (double)sum_waiting_time / (double) n);
    printf("Total Turnaround Time   : %-2d\n", sum_turnaround_time);
    printf("Average Turnaround Time : %-2.2lf\n", (double)sum_turnaround_time / (double) n);

    // print Gantt chart
    puts(""); // Empty line
    puts("          GANTT CHART          ");
    puts("          ***********          ");
    print_gantt_chart(p, n);
    return 0;
}


void print_table(Process p[], int n)
{
    int i;

    puts("+-----+------------+--------------+-----------------+");
    puts("| PID | Burst Time | Waiting Time | Turnaround Time |");
    puts("+-----+------------+--------------+-----------------+");

    for(i=0; i<n; i++) {
        printf("| %2d  |     %2d     |      %2d      |        %2d       |\n"
              , p[i].pid, p[i].burst_time, p[i].waiting_time, p[i].turnaround_time );
        puts("+-----+------------+--------------+-----------------+");
    }

}
```

```
void print_gantt_chart(Process p[], int n)
{
    int i, j;
    // print top bar
    printf(" ");
    for(i=0; i<n; i++) {
        for(j=0; j<p[i].burst_time; j++) printf("--");
        printf(" ");
    }
    printf("\n|");

    // printing process id in the middle
    for(i=0; i<n; i++) {
        for(j=0; j<p[i].burst_time - 1; j++) printf(" ");
        printf("P%d", p[i].pid);
        for(j=0; j<p[i].burst_time - 1; j++) printf(" ");
        printf("|");
    }
    printf("\n ");
    // printing bottom bar
    for(i=0; i<n; i++) {
        for(j=0; j<p[i].burst_time; j++) printf("--");
        printf(" ");
    }
    printf("\n");

    // printing the time line
    printf("0");
    for(i=0; i<n; i++) {
        for(j=0; j<p[i].burst_time; j++) printf("  ");
        if(p[i].turnaround_time > 9) printf("\b"); // backspace : remove 1 space
        printf("%d", p[i].turnaround_time);

    }
    printf("\n");

}
```

OUTPUT:

input

```
Enter total number of process: 4
Enter burst time for each process:
P[1] : 2
P[2] : 4
P[3] : 3
P[4] : 5

+-----+------------+--------------+-----------------+
| PID | Burst Time | Waiting Time | Turnaround Time |
+-----+------------+--------------+-----------------+
|  1  |      2     |       0      |         2       |
+-----+------------+--------------+-----------------+
|  2  |      4     |       2      |         6       |
+-----+------------+--------------+-----------------+
|  3  |      3     |       6      |         9       |
+-----+------------+--------------+-----------------+
|  4  |      5     |       9      |        14       |
+-----+------------+--------------+-----------------+

Total Waiting Time      : 17
Average Waiting Time    : 4.25
Total Turnaround Time   : 31
Average Turnaround Time : 7.75

          GANTT CHART
          ***********
 ---- -------- ------ ----------
| P1 |   P2   | P3  |    P4    |
 ---- -------- ------ ----------
0    2        6     9         14


...Program finished with exit code 0
```

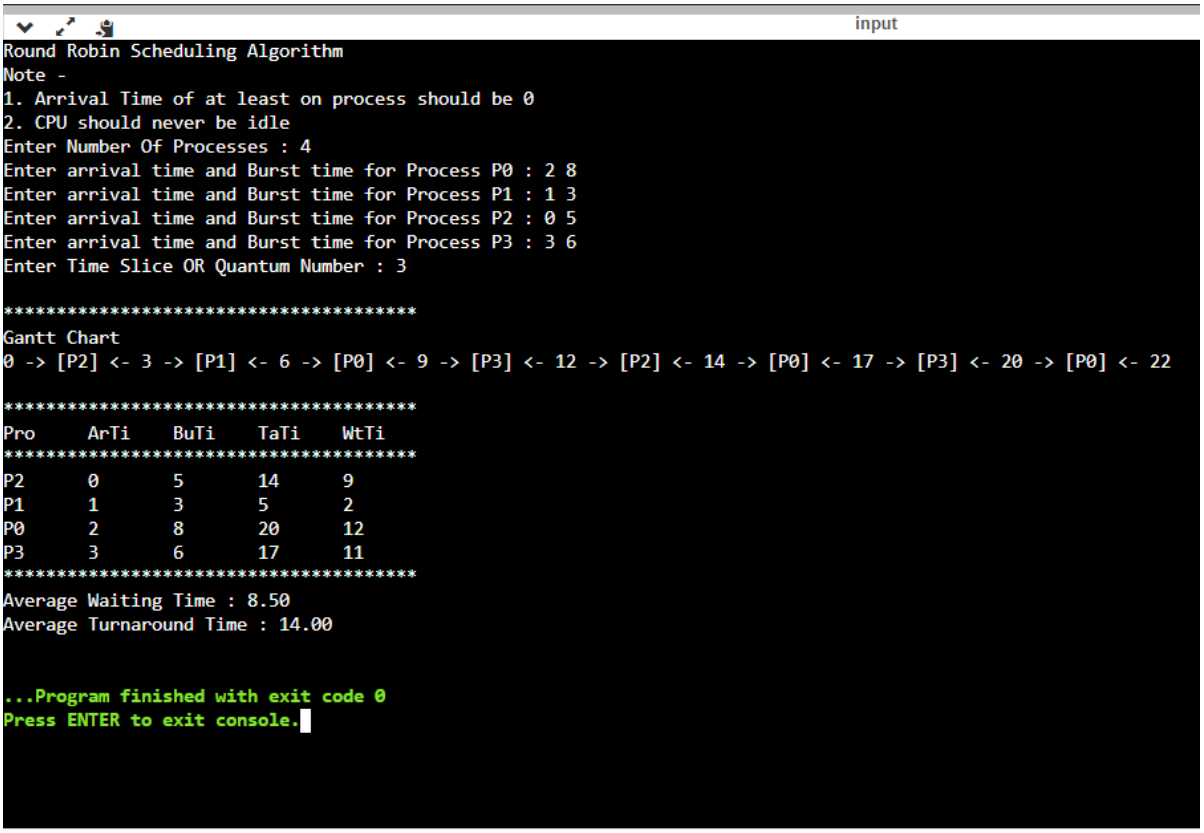
2. RR-Round Robin Scheduling Algorithm-Preemptive

CODE(C):

```
#include<stdio.h>

struct times
{
    int p,art,but,wtt,tat,rnt;
};


void sortart(struct times a[],int pro)
{
    int i,j;
    struct times temp;
    for(i=0;i<pro;i++)
    {
        for(j=i+1;j<pro;j++)
        {
            if(a[i].art > a[j].art)
            {
                temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
        }
    }
    return;
}

int main()
{
    int i,j,pro,time,remain,flag=0,ts;
    struct times a[100];
    float avgwt=0,avgtt=0;
    printf("Round Robin Scheduling Algorithm\n");
    printf("Note -\n1. Arrival Time of at least on process should be 0\n2. CPU should never be idle\n");
    printf("Enter Number Of Processes : ");
    scanf("%d",&pro);
    remain=pro;
    for(i=0;i<pro;i++)
    {
        printf("Enter arrival time and Burst time for Process P%d : ",i);
        scanf("%d%d",&a[i].art,&a[i].but);
        a[i].p = i;
        a[i].rnt = a[i].but;
    }
    sortart(a,pro);
    printf("Enter Time Slice OR Quantum Number : ");
    scanf("%d",&ts);
```

```
    printf("\n**************************************\n");
    printf("Gantt Chart\n");
    printf("0");
    for(time=0,i=0;remain!=0;)
    {
        if(a[i].rnt<=ts && a[i].rnt>0)
        {
            time = time + a[i].rnt;
            printf(" -> [P%d] <- %d",a[i].p,time);
            a[i].rnt=0;
            flag=1;
        }
        else if(a[i].rnt > 0)
        {
            a[i].rnt = a[i].rnt - ts;
            time = time + ts;
            printf(" -> [P%d] <- %d",a[i].p,time);
        }
        if(a[i].rnt==0 && flag==1)
        {
            remain--;
            a[i].tat = time-a[i].art;
            a[i].wtt = time-a[i].art-a[i].but;
            avgwt = avgwt + time-a[i].art-a[i].but;
            avgtt = avgtt + time-a[i].art;
            flag=0;
        }
        if(i==pro-1)
            i=0;
        else if(a[i+1].art <= time)
            i++;
        else
            i=0;
    }
    printf("\n\n");
    printf("**************************************\n");
    printf("Pro\tArTi\tBuTi\tTaTi\tWtTi\n");
    printf("**************************************\n");
    for(i=0;i<pro;i++)
    {
        printf("P%d\t%d\t%d\t%d\t%d\n",a[i].p,a[i].art,a[i].but,a[i].tat,a[i].wtt);
    }
    printf("**************************************\n");
    avgwt = avgwt/pro;
    avgtt = avgtt/pro;
    printf("Average Waiting Time : %.2f\n",avgwt);
    printf("Average Turnaround Time : %.2f\n",avgtt);
    return 0;
}
```

OUTPUT:



Outcome: CO2: Demonstrate use of inter process communication

Conclusion: We learnt and implemented one Preemptive and one Non-Preemptive scheduling algorithms and calculated the average waiting time as well as the average turn-around time.Results uploaded on drive along with the outcome.

Grade: AA/AB/BB/BC/CC/CD/DD

Signature of faculty in-charge with date

References:
Books/ Journals/ Websites:

1 Silberschatz A., Galvin P., Gagne G, “Operating Systems Concepts”, VIIIth Edition, Wiley, 2011.