ARUN M

76

R4B

Cycle 3
CPU Scheduling Algorithms

Experiment No. 3.1

Aim : Implementation of FCFS and SJF scheduling algorithm

(1) Write a Menu driven program to implement FCFS and SJF CPU scheduling algorithm (Non Preemeptive). Read burst time from the user. Display the Gantt chart and compute the following.

a) Waiting time of each process.
b) Average waiting Time.
c) Turn Around Time of each process.
d) Average Turn Around Time.
e) Throughput.

Program:

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

#define MAX_PROCESSES 20

int n, BT1[MAX_PROCESSES], BT2[MAX_PROCESSES], P1[MAX_PROCESSES],
P2[MAX_PROCESSES];
int WT[MAX_PROCESSES], TAT[MAX_PROCESSES];
float avg_WT, avg_TAT, t, tp;

// Function prototypes
void FCFS();
void SJF();

int main() {
   int choice;

   printf("Enter the number of processes: ");
   scanf("%d", &n);

   for (int i = 0; i < n; i++) {
      printf("Enter the burst time of process %d = ", i + 1);
      scanf("%d", &BT1[i]);
      BT2[i] = BT1[i];
      P1[i] = i + 1;
      P2[i] = i + 1;
      t = t + BT1[i];
   }
```

```
    while (1) {
        printf("\nMenu\n1. FCFS\n2. SJF\n3. Exit\n");
        printf("Enter the operation you want to perform: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                FCFS();
                break;
            case 2:
                SJF();
                break;
            case 3:
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}

void FCFS() {
    avg_WT = 0;
    avg_TAT = 0;

    WT[0] = 0;
    for (int i = 1; i < n; i++) {
        WT[i] = WT[i - 1] + BT1[i - 1];
        avg_WT += WT[i];
    }
    avg_WT /= n;

    for (int i = 0; i < n; i++) {
        TAT[i] = WT[i] + BT1[i];
        avg_TAT += TAT[i];
    }
    avg_TAT /= n;
    tp = n / t;

    printf("\nFCFS Scheduling:\n");
    printf("\nGantt chart\n|");
    for (int i = 0; i < n; i++) {
        printf("   P%d\t|", P1[i]);
    }
    printf("\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t", WT[i]);
    }
    printf("%d\n", TAT[n - 1]);
    printf("\nProcess\tBT\tWT\tTAT\n");
    for (int i = 0; i < n; i++) {
        printf("P%d\t%d\t%d\t%d\n", P1[i], BT1[i], WT[i], TAT[i]);
    }
    printf("Average Waiting Time = %f\n", avg_WT);
    printf("Average Turnaround Time = %f\n", avg_TAT);
    printf("Throughput = %f\n", tp);
}

void SJF() {
    int BTtemp, Ptemp;

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (BT2[j] > BT2[j + 1]) {
                BTtemp = BT2[j];
                BT2[j] = BT2[j + 1];
                BT2[j + 1] = BTtemp;

                Ptemp = P2[j];
                P2[j] = P2[j + 1];
                P2[j + 1] = Ptemp;
            }
        }
    }

    avg_WT = 0;
    avg_TAT = 0;
    WT[0] = 0;
    for (int i = 1; i < n; i++) {
        WT[i] = WT[i - 1] + BT2[i - 1];
        avg_WT += WT[i];
    }
    avg_WT /= n;

    for (int i = 0; i < n; i++) {
        TAT[i] = WT[i] + BT2[i];
        avg_TAT += TAT[i];
    }
    avg_TAT /= n;
    tp = n / t;

    printf("\nSJF Scheduling:\n");
    printf("\nGantt chart\n|");
    for (int i = 0; i < n; i++) {
        printf("   P%d\t|", P2[i]);
    }
    printf("\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t", WT[i]);
    }
    printf("%d\n", TAT[n - 1]);
    printf("\nProcess\tBT\tWT\tTAT\n");
    for (int i = 0; i < n; i++) {
        printf("P%d\t%d\t%d\t%d\n", P2[i], BT2[i], WT[i], TAT[i]);
    }
    printf("Average Waiting Time = %f\n", avg_WT);
    printf("Average Turnaround Time = %f\n", avg_TAT);
    printf("Throughput = %f\n", tp);
}
```

Output:

```
Enter the number of processes: 5
Enter the burst time of process 1 = 2
Enter the burst time of process 2 = 3
Enter the burst time of process 3 = 4
Enter the burst time of process 4 = 1
Enter the burst time of process 5 = 6

Menu
1. FCFS
2. SJF
3. Exit
Enter the operation you want to perform: 1

FCFS Scheduling:

Gantt chart
|  P1   |  P2   |  P3   |  P4   |  P5   |
0   2   5       9       10      16

Process         BT      WT      TAT
P1  2   0       2
P2  3   2       5
P3  4   5       9
P4  1   9       10
P5  6   10      16
Average Waiting Time = 5.200000
Average Turnaround Time = 8.400000
Throughput = 0.312500

Menu
1. FCFS
2. SJF
3. Exit
Enter the operation you want to perform: 2

SJF Scheduling:

Gantt chart
|  P4   |  P1   |  P2   |  P3   |  P5   |
0   1   3       6       10      16

Process         BT      WT      TAT
P4  1   0       1
P1  2   1       3
P2  3   3       6
P3  4   6       10
P5  6   10      16
Average Waiting Time = 4.000000
Average Turnaround Time = 7.200000
Throughput = 0.312500

Menu
1. FCFS
2. SJF
3. Exit
Enter the operation you want to perform: 3


=== Code Execution Successful ===
```

Experiment No. 3.2

Aim : Implementation of SRTF scheduling algorithm

(2) Write a Program to implement SJF CPU Scheduling algorithm (both preemptive and non-preemptive) and calculate Average Waiting Time, Average Turn Around Time , Average Response Time and Throughput of the system.

Program:

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

void main()
{
        int
i,n,P[10],BT[10],AT[10],WT[10],TAT[10],temp[10],flag=0,x,small,count=0;
        float avg_WT,avg_TAT,wt=0,tat=0,time,end;

        printf("Enter the number of processes:");
        scanf("%d",&n);

        for(i=0;i<n;i++)
        {
           printf("\nEnter the arrival time of process %d = ",i+1);
           scanf("%d",&AT[i]);
        }

        for(i=0;i<n;i++)
        {
           printf("\nEnter the burst time of process %d = ",i+1);
```

```
   scanf("%d",&BT[i]);

   P[i]=i+1;
   temp[i]=BT[i];
}

printf("\nProcess\tAT\tBT\n");
for(i=0;i<n;i++)
{
   printf("P%d\t%d\t%d\n",P[i],AT[i],BT[i]);
}

printf("\nPreemptive:\n");

BT[9]=9999;
printf("\nGantt chart:\n|");
for(time=0;count!=n;time++)
{
   small=9;
   for(i=0;i<n;i++)
   {
            if(AT[i]<=time && BT[i]<BT[small] && BT[i]>0)
            {
                     small=i;
                     if(flag==0)
                     {
                              printf(" %d |", AT[small]);
                              printf(" P%d ",small+1);
                              x=small;
                     }
                     flag=1;
            }
   }
   BT[small]--;
   if(BT[small]==0)
   {
            count++;
            end=time+1;
            wt=wt+end-AT[small]-temp[small];
            tat=tat+end-AT[small];
   }
   if(small!=x)
   {
            printf("| %d |",(int)time);
            printf(" P%d ",small+1);
            x=small;
   }
}
printf("| %d |\n",(int)end);

avg_WT=wt/n;
avg_TAT=tat/n;

printf("Average Waiting Time = %f\n",avg_WT);
printf("Average Turnaround Time = %f\n",avg_TAT);
```

```
        printf("Throughput = %f\n",n/end);
}
```

Output:

```
Enter the number of processes:4

Enter the arrival time of process 1 = 1

Enter the arrival time of process 2 = 2

Enter the arrival time of process 3 = 3

Enter the arrival time of process 4 = 4

Enter the burst time of process 1 = 3

Enter the burst time of process 2 = 2

Enter the burst time of process 3 = 4

Enter the burst time of process 4 = 1

Process         AT      BT
P1   1  3
P2   2  2
P3   3  4
P4   4  1

Preemptive:

Gantt chart:
|| 0 | P10  1 | P1 | 4 | P4 | 5 | P2 | 7 | P3 | 11 |
Average Waiting Time = 1.750000
Average Turnaround Time = 4.250000
Throughput = 0.363636


=== Code Execution Successful ===
```

Experiment No. 3.3

Aim : Implementation of priority scheduling algorithm

(3) Write a program to implement Priority Scheduling algorithm (Preemptive and Non-Preemptive). Read burst time, priority, arrival time and display the following.

a) Waiting time of each process.
b) Average waiting Time.
c) Turn Around Time of each process.
d) Average Turn Around Time.
e) Throughput and Gannt chart

Program:

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

int arrival[10], burst_time[10], burst_time_copy[10], temp[10], priority[10], wait_time[10],
turnaround_time[10];
int i, p, num_processes, x, k, prev, count, flag;
float avg_wait_time, avg_turnaround_time, current_time, end_time, wait_sum,
turnaround_sum, current_wait, current_turnaround;

void preemptive() {
    current_wait = 0;
    current_turnaround = 0;
    count = 0;
    flag = 0;
    printf("\nGantt Chart\n");
    for (current_time = 0; count != num_processes; current_time++) {
        p = 9;
        for (i = 0; i < num_processes; i++) {
            if (arrival[i] < current_time && priority[i] < priority[p] && burst_time[i] > 0) {
                p = i;
            }
        }
        if (p == 9) {
            continue;
        }
        if (flag == 0) {
            printf(" %d", arrival[p]);
            printf(" P%d", p + 1);
            x = p;
        }
        flag = 1;
        burst_time[p]--;
        if (burst_time[p] == 0) {
            count++;
            end_time = current_time + 1;
            current_wait += end_time - arrival[p] - temp[p];
            wait_time[p] += end_time - arrival[p] - temp[p];
            current_turnaround += end_time - arrival[p];
            turnaround_time[p] += end_time - arrival[p];
        }
        if (p != x) {
            printf(" %d", (int)current_time);
            printf(" P%d", p + 1);
            x = p;
        }
    }
    printf(" %d\n", (int)end_time);
    printf("Process\tBT\tWT\tTAT\n");
    for (i = 0; i < num_processes; i++) {
        printf("\nP%d\t%d\t%d", i + 1, wait_time[i], turnaround_time[i]);
    }
    avg_wait_time = current_wait / num_processes;
    avg_turnaround_time = current_turnaround / num_processes;
    printf("\nAverage waiting time = %f\n", avg_wait_time);
    printf("Average turnaround time = %f\n", avg_turnaround_time);
    printf("Throughput = %f\n", num_processes / end_time);
}

void nonpreemptive() {
    current_wait = 0;
    current_turnaround = 0;
    count = 0;
    flag = 0;
    printf("\nGantt Chart\n");
    for (current_time = 0; count != num_processes; current_time++) {
        p = 9;
        for (i = 0; i < num_processes; i++) {
            if (arrival[i] <= current_time && priority[i] < priority[p] && burst_time_copy[i] >
0) {
                p = i;
            }
        }
        if (p == 9) {
            continue;
        }
        if (flag == 0) {
            printf("%d", arrival[p]);
        }
        flag = 1;
        while (burst_time_copy[p] > 0) {
            burst_time_copy[p]--;
            current_time++;
            for (k = 0; k < num_processes; k++) {
                if (burst_time_copy[k] != 0 && k != p) {
                    wait_time[k]++;
                }
            }
        }
        if (burst_time_copy[p] == 0) {
            current_time--;
            count++;
            end_time = current_time + 1;
            current_wait += end_time - arrival[p] - temp[p];
            current_turnaround += end_time - arrival[p];
        }
        printf(" P%d", p + 1);
        printf(" %d", (int)current_time + 1);
    }
    printf("\nProcess\tBT\tWT\tTAT\n");
    for (i = 0; i < num_processes; i++) {
        printf("\nP%d\t%d\t%d\t%d", i + 1, burst_time[i], wait_time[i] - arrival[i], wait_time[i]
- arrival[i] + temp[i]);
    }
    avg_wait_time = current_wait / num_processes;
    avg_turnaround_time = current_turnaround / num_processes;
    printf("Average waiting time = %f\n", avg_wait_time);
    printf("Average turnaround time = %f\n", avg_turnaround_time);
    printf("Throughput = %f\n", num_processes / end_time);
}
```

```
int main() {
    printf("Enter the number of processes: ");
    scanf("%d", &num_processes);
    for (i = 0; i < num_processes; i++) {
        printf("Process %d\n", i + 1);
        printf("Enter Arrival time: ");
        scanf("%d", &arrival[i]);
        printf("Enter burst time: ");
        scanf("%d", &burst_time[i]);
        printf("Enter priority: ");
        scanf("%d", &priority[i]);
        temp[i] = burst_time[i];
        burst_time_copy[i] = burst_time[i];
        wait_time[i] = 0;
        turnaround_time[i] = 0;
    }
    priority[9] = 9999;
    while (1) {
        int choice;
        printf("\nMenu\n1. Preemptive\n2. Non-Preemptive\n3. Exit\n");
        printf("Enter the operation you want to perform: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                preemptive();
                break;
            case 2:
                nonpreemptive();
                break;
            case 3:
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }
    return 0;
}
```

Output:

```
/tmp/osCnGO2Uep.o
Enter the number of processes: 4
Process 1
Enter Arrival time: 2
Enter burst time: 3
Enter priority: 2
Process 2
Enter Arrival time: 3
Enter burst time: 2
Enter priority: 4
Process 3
Enter Arrival time: 1
Enter burst time: 2
Enter priority: 2
Process 4
```

```
Enter Arrival time: 3
Enter burst time: 5
Enter priority: 2

Menu
1. Preemptive
2. Non-Preemptive
3. Exit
Enter the operation you want to perform: 1

Gantt Chart
 1 P3 3 P1 6 P3 7 P4 12 P2 14
Process BT      WT      TAT

P1      1       4
P2      9       11
P3      4       6
P4      4       9
Average waiting time = 4.500000
Average turnaround time = 7.500000
Throughput = 0.285714

Menu
1. Preemptive
2. Non-Preemptive
3. Exit
Enter the operation you want to perform: 2

Gantt Chart
1 P3 3 P1 6 P4 11 P2 13
Process BT      WT      TAT

P1      0       1       4
P2      0       16      18
P3      0       3       5
P4      0       6       11Average waiting time = 3.000000
Average turnaround time = 6.000000
Throughput = 0.307692

Menu
1. Preemptive
2. Non-Preemptive
3. Exit
Enter the operation you want to perform: 3


=== Code Execution Successful ===
```

Experiment No. 3.4

Aim : Implementation of round robin scheduling algorithm

(4) Write a Program to implement Round Robin Scheduling algorithm. Implement the program as a menu driven on the basis of time quantum (possible values of time quantums are : 2ms, 4ms, 5ms, 8ms and 10 ms). Display the Gantt chart and calculate Average Waiting Time, Average Turn Around Time and Throughput of the system in each case.

Program:

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

struct Process {
    int id;
    int burst;
    int waiting;
    int turnaround;
};

void calculateMetrics(struct Process* processes, int n, int quantum) {
    struct Process* queue[n];
    int front = 0, rear = -1;
    int total_time = 0;
    int time_quantum = quantum;

    for (int i = 0; i < n; i++) {
        queue[++rear] = &processes[i];
    }

    while (front <= rear) {
        struct Process* p = queue[front++];

        if (p->burst <= time_quantum) {
            total_time += p->burst;
            p->turnaround = total_time;
            p->waiting = total_time - p->burst;
            for (int i = front; i <= rear; i++) {
                queue[i-1] = queue[i];
            }
            rear--;
        } else {
            total_time += time_quantum;
            p->burst -= time_quantum;
            queue[++rear] = p;
        }
    }

    float total_waiting = 0, total_turnaround = 0;
    for (int i = 0; i < n; i++) {
        total_waiting += processes[i].waiting;
        total_turnaround += processes[i].turnaround;
    }
    float avg_waiting = total_waiting / n;
    float avg_turnaround = total_turnaround / n;
    float throughput = n / (float) total_time;

    printf("Gantt Chart:\n");
    printf("0");
    for (int i = 0; i < n; i++) {
        printf("  P%d  %d", processes[i].id, processes[i].turnaround);
    }
    printf("\n");

    printf("Average Waiting Time: %f\n", avg_waiting);
    printf("Average Turnaround Time: %f\n", avg_turnaround);
    printf("Throughput: %f\n", throughput);
}

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

    struct Process processes[n];
    for (int i = 0; i < n; i++) {
        printf("Enter burst time for process %d: ", i+1);
        scanf("%d", &processes[i].burst);
        processes[i].id = i+1;
        processes[i].waiting = 0;
        processes[i].turnaround = 0;
    }

    int time_quantum_values[] = {2, 4, 5, 8, 10};
    for (int i = 0; i < 5; i++) {
        printf("\nTime Quantum = %d ms\n", time_quantum_values[i]);
        calculateMetrics(processes, n, time_quantum_values[i]);
    }

    return 0;
}
```

Output:

```
/tmp/ZmGj34Mjh3.o
Enter number of processes: 4
Enter burst time for process 1: 2
Enter burst time for process 2: 3
Enter burst time for process 3: 4
Enter burst time for process 4: 1

Time Quantum = 2 ms
Gantt Chart:
0  P1  2  P2  0  P3  0  P4  5
Average Waiting Time: 1.000000
Average Turnaround Time: 1.750000
Throughput: 0.800000

Time Quantum = 4 ms
Gantt Chart:
0  P1  2  P2  0  P3  4  P4  5
Average Waiting Time: 1.500000
Average Turnaround Time: 2.750000
```

```
Throughput: 1.000000

Time Quantum = 5 ms
Gantt Chart:
0  P1  2  P2  0  P3  4  P4  5
Average Waiting Time: 1.500000
Average Turnaround Time: 2.750000
Throughput: 1.000000

Time Quantum = 8 ms
Gantt Chart:
0  P1  2  P2  0  P3  4  P4  5
Average Waiting Time: 1.500000
Average Turnaround Time: 2.750000
Throughput: 1.000000

Time Quantum = 10 ms
Gantt Chart:
0  P1  2  P2  0  P3  4  P4  5
Average Waiting Time: 1.500000
Average Turnaround Time: 2.750000
Throughput: 1.000000


=== Code Execution Successful ===
```