ST. FRANCIS INSTITUE OF TECHNOLOGY

MT. POINSUR, BORIVALI (W), MUMBAI

LAB MANUAL

ON

OPERATING SYSTEM LAB

SE-CMPN-A & B / IV

Academic Year: 2020-2021

Prepared By:

Ms. Nidhi Gaur

Asst. Professor in CMPN Department

Experiment No. 1

Aim: Write a program to implement FCFS and SJF CPU Scheduling algorithm

Theory: For FCFS scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times. The scheduling is performed on the basis of arrival time of the processes irrespective of their other parameters. Each process will be executed according to its arrival time. Calculate the waiting time and turnaround time of each of the processes accordingly.

Algorithm:

1. Start
2. Read the number of processes to be inserted
3. Read the Burst times and Arrival time of processes
4. Calculate start time and finish time of every process
5. Calculate the waiting time of each process

   wt[i]=start[i]-arr[i];
6. Calculate the turnaround time of each process

   tat[i]=finish[i]-arr[i];
7. Calculate the average waiting time and average turnaround time.
8. Display the values
9. Stop

Example:

Process	Burst Time
P1	24
P2	03
P3	03

Suppose that the processes arrive in the order: P1 , P2 , P3

The Gantt Chart for the schedule is:

Waiting time for P1 = 0; P2 = 24; P3 = 27

Average waiting time: (0 + 24 + 27)/3 = 17

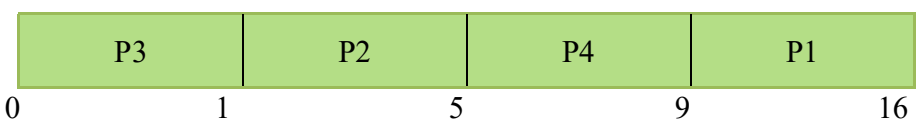
Theory: For SJF scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times. Arrange all the jobs in order with respect to their burst times. There may be two jobs in queue with the same execution time, and then FCFS approach is to be performed. Each process will be executed according to the length of its burst time. Then calculate the waiting time and turnaround time of each of the processes accordingly.

Algorithm:

1. Start
2. Read the number of processes to be inserted
3. Read the Burst times and Arrival time of processes
4. Arrange processes in ascending order w.r.t. their burst time
5. Calculate waiting time and turnaround time of each process
6. Calculate the average waiting time and average turnaround time.
7. Display the values
8. Stop

SOLVE Example:

Process	Burst Time
P1	7
P2	4
P3	1
P4	4



Average Waiting Time: 3.75

CODE:

A. FCFS Algorithm

```
#include<stdio.h>
int main()
{
	int n,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;
	printf("Enter total number of processes(maximum 20):");
	scanf("%d",&n);
	printf("\nEnter Process Burst Time\n");
	for(i=0;i<n;i++)
	{
		printf("P[%d]:",i+1);
		scanf("%d",&bt[i]);
	}
	wt[0]=0;
	for(i=1;i<n;i++)
	{
		wt[i]=0;
		for(j=0;j<i;j++)
		wt[i]+=bt[j];
	}
	printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time");
	for(i=0;i<n;i++)
	{
		tat[i]=bt[i]+wt[i];
		avwt+=wt[i];
		avtat+=tat[i];
		printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,bt[i],wt[i],tat[i]);
	}
	avwt/=i;
	avtat/=i;
	printf("\n\nAverage Waiting Time:%d",avwt);
	printf("\nAverage Turnaround Time:%d",avtat);
	return 0;
}
```

OUTPUT:

```
Enter total number of processes(maximum 20):4

Enter Process Burst Time
P[1]:4
P[2]:6
P[3]:2
P[4]:8

Process         Burst Time      Waiting Time    Turnaround Time
P[1]            4               0               4
P[2]            6               4               10
P[3]            2               10              12
P[4]            8               12              20

Average Waiting Time:6
Average Turnaround Time:11
Press Enter to return to Quincy...
```

B. SJF Algorithm

CODE:

```
#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;
	}
	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("\np%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=%f",avg_wt);
        printf("\nAverage Turnaround Time=%f\n",avg_tat);
}
```

OUTPUT:

```
Enter number of process:4

Enter Burst Time:
p1:7
p2:4
p3:1
p4:4

Process     Burst Time          Waiting Time    Turnaround Time
p3                1                   0                   1
p2                4                   1                   5
p4                4                   5                   9
p1                7                   9                   16

Average Waiting Time=3.750000
Average Turnaround Time=7.750000

Press Enter to return to Quincy...
```

Viva questions:

Q1: What is drawback of FCFS algorithm?

Q2: SJF eliminates drawback of FCFS. Is it true or false? Solve one numerical justifying your answer.

O·S Experiment 1 (42)

Q1) Drawback of FCFS Algorithm

① Process with less time execution that is waiting time is quiet long.

② Favours CPU bound process and I/O bound process

③ First process will get CPU first and other processes can get CPU only after current process has finished execution. If first process has large burst time and others have less than other processes will have to wait longer resulting in more average waiting time called convoy effect

④ Results in low CPU and device utilization

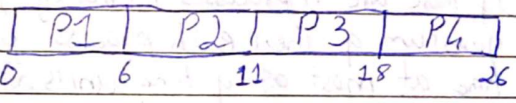
⑤ Difficult for time sharing system.

Q2)

Eg

Process	Arrival Time	Execution Time
P1	0	6
P2	3	5
P3	4	8
P4	6	7

GANTT CHART:



Process	Arrival Time	TAT	Waiting Time
P1	0	6	0-0=0
P2	3	8	6-3=3
P3	4	22	18-4=14
P4	6	18	11-6=5

Average TAT = (6+8+22+18)/4 = 12ms

Average Waiting Time = (0+3+14+5)/4 = 5.5ms

CONCLUSION:

The algorithms FCFS and SJF are used to find the averages of turnaround time and waiting time. Gantt chart clears most of the concept to find the TAT and WT. The drawbacks given by FCFS are cleared by SJF.

Experiment : 02

AIM :- To write a program to implement the ROUND ROBIN Scheduling using C

PROBLEM DESCRIPTION:

In this algorithm we are assigning some time slice .The process is allocated according to the time slice, if the process service time is less than the time slice then process itself will release the CPU voluntarily. The scheduler will then proceed to the next process in the ready queue. If the CPU burst of the currently running process is longer than time quantum, the timer will go off and will cause an interrupt to the operating system .A context switch will be executed and the process will be put at the tail of the ready queue.

ALGORITHM:

1. Start
2. Declare the array size
3. Read the number of processes to be inserted
4. Read the burst times of the processes
5. Read the Time Quantum
6. If the burst time of a process is greater than time Quantum then subtract time quantum form the burst time Else Assign the burst time to time quantum.
7. Calculate the average waiting time and turnaround time of the processes.
8. Display the values
9. Stop

Example Of Round Robin Scheduling

Process	Arrival Time	Burst Time
P0	0	9
P1	1	5
P2	2	3
P4	3	4

Calculate Avg TAT and Avg WT with Time Quantum =2 ms

Code:

```
#include<stdio.h>
int main()
{
	int i, limit, total = 0, x, counter = 0, time_quantum;
	int wait_time = 0, turnaround_time = 0, arrival_time[10], burst_time[10],
	temp[10];
	float average_wait_time, average_turnaround_time;
	printf("\nEnter Total Number of Processes:\t");
	scanf("%d", &limit);
	x = limit;
	for(i = 0; i < limit; i++)
	{
		printf("\nEnter Details of Process[%d]\n", i + 1);
		printf("Arrival Time:\t");
		scanf("%d", &arrival_time[i]);
		printf("Burst Time:\t");
		scanf("%d", &burst_time[i]);
		temp[i] = burst_time[i];
	}
	printf("\nEnter Time Quantum:\t");
	scanf("%d", &time_quantum);
	printf("\nProcess ID\t\tBurst Time\t Turnaround Time\t Waiting Time\n");
	for(total = 0, i = 0; x != 0;)
	{
		if(temp[i] <= time_quantum && temp[i] > 0)
		{
			total = total + temp[i];
			temp[i] = 0;
			counter = 1;
		}
		else if(temp[i] > 0)
		{
			Temp[i] = temp[i] - time_quantum;
			total = total + time_quantum;
```

```
            }
            if(temp[i] == 0 && counter == 1)
            {
                x--;
                printf("\nProcess[%d]\t\t%d\t\t %d\t\t\t %d", i + 1, burst_time[i],
                total - arrival_time[i], total - arrival_time[i] - burst_time[i]);
                wait_time = wait_time + total - arrival_time[i] - burst_time[i];
                turnaround_time = turnaround_time + total - arrival_time[i];
                counter = 0;
            }
            if(i == limit - 1)
            {
                i = 0;
            }
            else if(arrival_time[i + 1] <= total)
            {
                i++;
            }
            else
            {
                i = 0;
            }
        }
        average_wait_time = wait_time * 1.0 / limit;
        average_turnaround_time = turnaround_time * 1.0 / limit;
        printf("\n\nAverage Waiting Time:\t%f", average_wait_time);
        printf("\nAvg Turnaround Time:\t%f\n", average_turnaround_time);
        return 0;
}
```

Output:

```
Enter Details of Process[1]
Arrival Time:   0
Burst Time:     9

Enter Details of Process[2]
Arrival Time:   1
Burst Time:     5

Enter Details of Process[3]
Arrival Time:   2
Burst Time:     3

Enter Details of Process[4]
Arrival Time:   3
Burst Time:     4

Enter Time Quantum:     2

Process ID              Burst Time      Turnaround Time         Waiting Time

Process[3]              3               11                      8
Process[4]              4               12                      8
Process[2]              5               17                      12
Process[1]              9               21                      12

Average Waiting Time:   10.000000
Avg Turnaround Time:    15.250000
```

Conclusion:

Hence, we see that Round Robin Scheduling Algorithm is very fair and no starvation occurs.

Viva question:

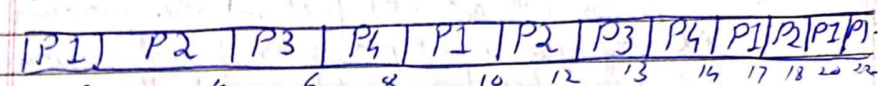
1. **Solve one numerical using Round Robin**
2. **What is the effect of time slice on context switching?**

O.S Experiment-2 (42)

Q1)

Process Queue	Burst Time	Arrival Time
P1	9	0
P2	5	1
P3	3	2
P4	4	3

$q = 2$ ms



Process	Arrival time	Burst Time	TAT	WT
P1	0	9	21	12
P2	1	5	17	12
P3	2	3	11	8
P4	3	4	12	8

Average TAT $= \frac{21+12+11+12}{4} = 15.25$ ms

Average WT $= \frac{12+12+8+8}{4} = 10$ ms

Q2.

→ If there are n processes in ready queue and time quantum q, then each process gets 1/n of CPU time at most of q time units at once. No process waits for more than $(n-1)q$ time units.

Performance: q is large FIFO → q is small more context switching

q must be large w.r.t context switch otherwise overhead is too high