

ASSIGNMENT NO.: 03

AIM:

Implement the C program for CPU Scheduling Algorithms: Shortest Job First (Preemptive) and Round Robin with different arrival time.

PREREQUISITE:

1. C Programming
2. Fundamentals of Data Structure

OBJECTIVE:

To study

- Preemptive and Non-Preemptive CPU scheduling
- Application and use of CPU scheduling Algorithm

THEORY:

Shortest Job First (Preemptive):

What is Shortest Job First?

This is an approach which considers the next CPU burst. Each process possess its next CPU burst. When CPU is available, the process having the smallest next CPU burst is allocated CPU.

It may happen that two or more processes have the same next CPU burst. Then which process to allocate will be decided as per FCFS scheduling.

Shortest job first(SJF) is a scheduling algorithm, that is used to schedule processes in an operating system. It is a very important topic in Scheduling when compared to round-robin and FCFS Scheduling.

There are two types of SJF

- Pre-emptive SJF
- Non-Preemptive SJF

These algorithms schedule processes in the order in which the shortest job is done first. It has a minimum average waiting time.

There are 3 factors to consider while solving SJF, they are

1. BURST Time

2. Average waiting time
3. Average turnaround time

Shortest Remaining Time First (SRTF) scheduling:

In the Shortest Remaining Time First (SRTF) scheduling algorithm, the process with the smallest amount of time remaining until completion is selected to execute. Since the currently executing process is the one with the shortest amount of time remaining by definition, and since that time should only reduce as execution progresses, processes will always run until they complete or a new process is added that requires a smaller amount of time.

Implementation Points:

1- Traverse until all process gets completely executed.

- a) Find process with minimum remaining time at every single time lap.
- b) Reduce its time by 1.
- c) Check if its remaining time becomes 0
- d) Increment the counter of process completion.
- e) Completion time of current process = current_time + 1;
- e) Calculate waiting time for each completed process.

$$wt[i] = \text{Completion time} - \text{arrival_time_burst_time}$$

f) Increment time lap by one.

2- Find turnaround time (waiting_time+burst_time).

Key Differences Between Preemptive and Non-Preemptive Scheduling:

1. In preemptive scheduling, the CPU is allocated to the processes for a limited time whereas, in Non-preemptive scheduling, the CPU is allocated to the process till it terminates or switches to the waiting state.
2. The executing process in preemptive scheduling is interrupted in the middle of execution when higher priority one comes whereas, the executing process in non-preemptive scheduling is not interrupted in the middle of execution and waits till its execution.

3. In Preemptive Scheduling, there is the overhead of switching the process from the ready state to running state, vice-versa and maintaining the ready queue. Whereas in the case of non-preemptive scheduling has no overhead of switching the process from running state to ready state.
4. In preemptive scheduling, if a high-priority process frequently arrives in the ready queue then the process with low priority has to wait for a long, and it may have to starve. , in the non-preemptive scheduling, if CPU is allocated to the process having a larger burst time then the processes with small burst time may have to starve.
5. Preemptive scheduling attains flexibility by allowing the critical processes to access the CPU as they arrive into the ready queue, no matter what process is executing currently. Non-preemptive scheduling is called rigid as even if a critical process enters the ready queue the process running CPU is not disturbed.
6. Preemptive Scheduling has to maintain the integrity of shared data that's why it is cost associative which is not the case with Non-preemptive Scheduling.

Shortest Job First Advantages and Disadvantages:

Advantages

- This algorithm is simple to implement.
- Does not depend on any priority of the process. The smallest burst time is the higher priority consideration.
- It provides good CPU utilization than FCFS (First Come First Search).
- Waiting time and turnaround time of each process is reduced, reducing the average waiting time and turn around the time of the system as compared to FCFS.

Disadvantages

- Waiting time of some processes still high due to the long burst time of the processes, in case of non-preemptive scheduling.
- In the case of non-preemptive scheduling, it may act as a uni-processing operating system.
- In the case of preemptive scheduling, context switch is required.
- And in preemptive scheduling, turnaround time may get increased.

Preemptive SJF Example:

Process	Duration	Order	Arrival Time
P1	9	1	0
P2	2	2	2

Shortest Job First (Preemptive) Code Implementation:**Code:**

```
#include <stdio.h>

int main()
{
    int arrival_time[10], burst_time[10], temp[10];

    int I, smallest, count = 0, time, limit;

    double wait_time = 0, turnaround_time = 0, end;

    float average_waiting_time, average_turnaround_time;

    printf("\nEnter the Total Number of Processes:t");

    scanf("%d", &limit);

    printf("\nEnter Details of %d Processesn", limit);

    for(I = 0; I < limit; i++)
    {
        printf("\nEnter Arrival Time:t");

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

        printf("\nEnter Burst Time:t");
```

```

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

temp[i] = burst_time[i];

}

burst_time[9] = 9999;

for(time = 0; count != limit; time++)
{
    smallest = 9;

    for(I = 0; I < limit; i++)
    {
        if(arrival_time[i] <= time && burst_time[i] < burst_time[smallest]
&& burst_time[i] > 0)
        {
            smallest = I;
        }
    }

    burst_time[smallest]--;

    if(burst_time[smallest] == 0)
    {
        count++;

        end = time + 1;

        wait_time = wait_time + end - arrival_time[smallest] -
temp[smallest];

        turnaround_time = turnaround_time + end - arrival_time[smallest];
    }
}

```

```

average_waiting_time = wait_time / limit;

average_turnaround_time = turnaround_time / limit;

printf("\nAverage Waiting Time:t%lf\n", average_waiting_time);

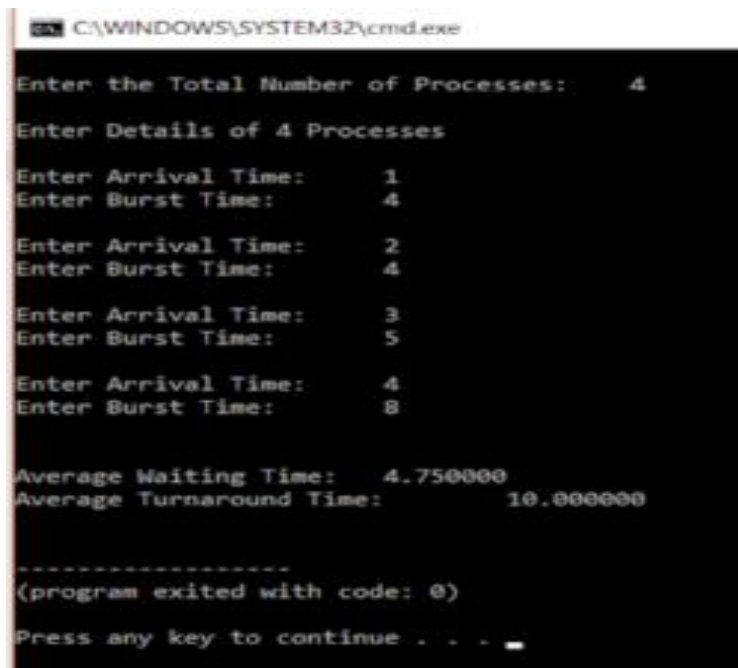
printf("Average Turnaround Time:t%lf\n", average_turnaround_time);

return 0;

}

```

Output



```

C:\WINDOWS\SYSTEM32\cmd.exe
Enter the Total Number of Processes: 4
Enter Details of 4 Processes
Enter Arrival Time: 1
Enter Burst Time: 4
Enter Arrival Time: 2
Enter Burst Time: 4
Enter Arrival Time: 3
Enter Burst Time: 5
Enter Arrival Time: 4
Enter Burst Time: 8

Average Waiting Time: 4.750000
Average Turnaround Time: 10.000000

-----
(program exited with code: 0)
Press any key to continue . . .

```

Round Robin with different arrival time

Round Robin Scheduling

- The name of this algorithm comes from the round-robin principle, where each person gets an equal share of something in turns.
- Each process is assigned a fixed time slot in a cyclic way. Time Quantum
- It is the oldest, simplest scheduling algorithm, which is mostly used for multitasking.
- In Round-robin scheduling, each ready task runs turn by turn only in a cyclic queue for a limited time slice.
- This algorithm also offers starvation free execution of processes.

Characteristics of Round Robin

- Round robin is a pre-emptive algorithm
- The CPU is shifted to the next process after fixed interval time, which is called time quantum/time slice.
- The process that is preempted is added to the end of the queue.
- Round robin is a hybrid model which is clock-driven
- Time slice should be minimum, which is assigned for a specific task that needs to be processed. However, it may differ OS to OS.
- It is a real time algorithm which responds to the event within a specific time limit.
- Round robin is one of the oldest, fairest, and easiest algorithm.
- Widely used scheduling method in traditional OS.

Advantages of Round Robin

- It does not face any starvation issues or convoy effect.
- Each process gets equal priority to the fair allocation of CPU.
- It deals with all process without any priority.
- It is easy to implement the CPU Scheduling algorithm.
- Each new process is added to the end of the ready queue as the next process's arrival time is reached.
- Each process is executed in circular order that shares a fixed time slot or quantum.
- Every process gets an opportunity in the round-robin scheduling algorithm to reschedule after a given quantum period.
- This scheduling method does not depend upon burst time. That's why it is easily implementable on the system.

Disadvantages of Round Robin

- If the time quantum is lower, it takes more time on context switching between the processes.
- It does not provide any special priority to execute the most important process.

- The waiting time of a large process is higher due to the short time slot.
- The performance of the algorithm depends on the time quantum.
- The response time of the process is higher due to large slices to time quantum.
- Getting a correct time slot or quantum is quite difficult for all processes in the round-robin algorithm.

Example of Round Robin #1

Process Queue	Burst Time
P1	4
P2	3
P3	5

Queue Representation based on burst on time

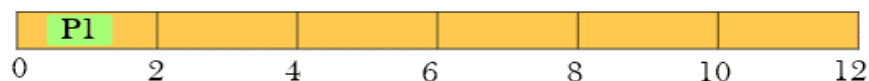
Now Time Slice = 2



- **Step 1)** The execution begins with process P1, which has burst time 4. Here, every process executes for 2 seconds. P2 and P3 are still in the waiting queue.



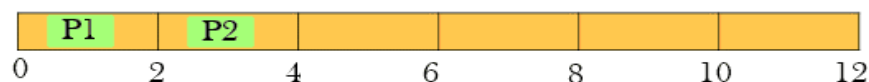
- Time Slice = 2



- **Step 2)** At time =2, P1 is added to the end of the Queue and P2 starts executing



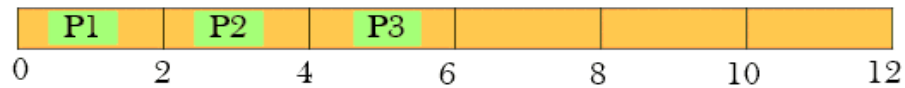
- Time Slice = 2



- **Step 3)** At time=4 , P2 is preempted and add at the end of the queue. P3 starts executing.



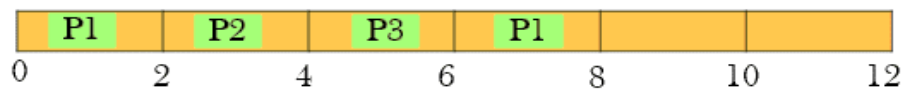
- Time Slice = 2



- **Step 4)** At time=6 , P3 is preempted and add at the end of the queue. P1 starts executing.



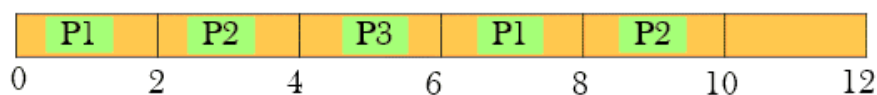
- Time Slice = 2



- **Step 5)** At time=8 , P1 has a burst time of 4. It has completed execution. P2 starts execution



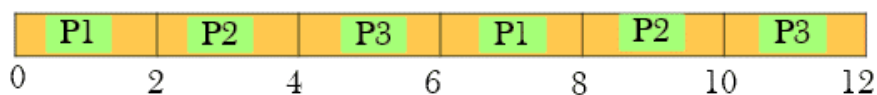
- Time Slice = 2



- **Step 6)** P2 has a burst time of 3. It has already executed for 2 interval. At time=9, P2 completes execution. Then, P3 starts execution till it completes.



- Time Slice = 2



- Now as all the process get equal time slice for execution let's calculate average waiting time(Service Time - Arrival Time).

Process Queue	Burst time	Waiting Time
P1	4	0+ 4= 4
P2	3	2+4= 6
P3	5	4+3= 7

Round Robin scheduling Code implementation

Code:

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

void main()
{
    // initialize the variable name
    int i, NOP, sum=0, count=0, y, quant, wt=0, tat=0, at[10], bt[10], temp[10];
    float avg_wt, avg_tat;
    printf(" Total number of process in the system: ");
    scanf("%d", &NOP);
    y = NOP; // Assign the number of process to variable y

    // Use for loop to enter the details of the process like Arrival time and the Burst
    Time
    for(i=0; i<NOP; i++)
    {
        printf("\n Enter the Arrival and Burst time of the Process[%d]\n", i+1);
        printf(" Arrival time is: \t"); // Accept arrival time
        scanf("%d", &at[i]);
        printf(" \nBurst time is: \t"); // Accept the Burst time
        scanf("%d", &bt[i]);
        temp[i] = bt[i]; // store the burst time in temp array
    }
    // Accept the Time qunat
    printf("Enter the Time Quantum for the process: \t");
    scanf("%d", &quant);
    // Display the process No, burst time, Turn Around Time and the waiting time
    printf("\n Process No \t\t Burst Time \t\t TAT \t\t Waiting Time ");
```

```

for(sum=0, i = 0; y!=0; )
{
if(temp[i] <= quant && temp[i] > 0) // define the conditions
{
    sum = sum + temp[i];
    temp[i] = 0;
    count=1;
}
else if(temp[i] > 0)
{
    temp[i] = temp[i] - quant;
    sum = sum + quant;
}
if(temp[i]==0 && count==1)
{
    y--; //decrement the process no.
    printf("\nProcess No[%d] \t\t %d\t\t\t\t %d\t\t\t\t %d", i+1, bt[i], sum-
at[i], sum-at[i]-bt[i]);
    wt = wt+sum-at[i]-bt[i];
    tat = tat+sum-at[i];
    count =0;
}
if(i==NOP-1)
{
    i=0;
}
else if(at[i+1]<=sum)
{
    i++;
}
}

```

```
else
{
    i=0;
}
}
}
```

Output:

