OS Lab Assignment 6

Krishna Pandey
U20CS110

Q1.)

Write a program for the simulation of
1. Shortest Job First (SJF)
2. Shortest Remaining Time First (SRTF) CPU scheduler.
3. Round Robin Scheduling with quantum 3 units.

Take n no of process from user with arrival time and burst time.
Arrival time and burst time should be generated randomly and compute Completion
Time, Turnaround (TAT) Time and Waiting Time.
Also show the count of context switching in all of the algorithms.

SJF

```
//Shortest Job First

#include <bits/stdc++.h>
using namespace std;
struct process
{
    int pid;
    int arrival_time;
    int burst_time;
    int start_time;
    int completion_time;
    int turnaround_time;
    int waiting_time;
    int response_time;
};
int main()
{

    int x;
    struct process p[100];
    float avg_turnaround_time;
    float avg_waiting_time;
    float avg_response_time;
    float cpu_utilization;
    int total_turnaround_time = 0;
    int total_waiting_time = 0;
    int total_response_time = 0;
    int total_idle_time = 0;
    float throughput;
    int burst_remaining[100];
int is_completed[100];
```

```
    memset(is_completed, 0, sizeof(is_completed));

    cout << setprecision(2) << fixed;

    cout << "Enter the number of processes: ";
    cin >> x;

    for (int i = 0; i < x; i++)
    {
        cout << "Enter arrival time ofthe process " << i + 1 << ":
";
        cin >> p[i].arrival_time;
        cout << "Enter burst time of the process " << i + 1 << ":
";
        cin >> p[i].burst_time;
        p[i].pid = i + 1;
        burst_remaining[i] = p[i].burst_time;
        cout << endl;
    }

    int current_time = 0;
    int completed = 0;
    int prev = 0;

    while (completed != x)
    {
        int idx = -1;
        int mn = 10000000;
        for (int i = 0; i < x; i++)
        {
            if (p[i].arrival_time <= current_time &&
is_completed[i] == 0)
            {
                if (burst_remaining[i] < mn)
                {
                    mn = burst_remaining[i];
                    idx = i;
                }
                if (burst_remaining[i] == mn)
                {
                    if (p[i].arrival_time < p[idx].arrival_time)
                    {
                        mn = burst_remaining[i];
                        idx = i;
                    }
                }
            }
        }

        if (idx != -1)
        {
            if (burst_remaining[idx] == p[idx].burst_time)
            {
                p[idx].start_time = current_time;
                total_idle_time += p[idx].start_time - prev;
            }
            burst_remaining[idx] -= 1;
            current_time++;
            prev = current_time;

            if (burst_remaining[idx] == 0)
            {
                p[idx].completion_time = current_time;
                p[idx].turnaround_time = p[idx].completion_time -
p[idx].arrival_time;
                p[idx].waiting_time = p[idx].turnaround_time -
p[idx].burst_time;
                p[idx].response_time = p[idx].start_time -
p[idx].arrival_time;

                total_turnaround_time += p[idx].turnaround_time;
                total_waiting_time += p[idx].waiting_time;
                total_response_time += p[idx].response_time;

                is_completed[idx] = 1;
                completed++;
            }
        }
        else
        {
            current_time++;
        }
    }

    int min_arrival_time = 10000000;
    int max_completion_time = -1;
    for (int i = 0; i < x; i++)
    {
        min_arrival_time = min(min_arrival_time,
p[i].arrival_time);
        max_completion_time = max(max_completion_time,
p[i].completion_time);
    }

    avg_turnaround_time = (float)total_turnaround_time / x;
    avg_waiting_time = (float)total_waiting_time / x;
    avg_response_time = (float)total_response_time / x;

    cout << endl
         << endl;

    cout << "P\t"
         << "AT\t"
         << "BT\t"
         << "ST\t"
         << "CT\t"
         << "TAT\t"
         << "WT\t"
         << "RT\t"
         << "\n"
         << endl;

    for (int i = 0; i < x; i++)
    {
        cout << p[i].pid << "\t" << p[i].arrival_time << "\t" <<
p[i].burst_time << "\t" << p[i].start_time << "\t" <<
p[i].completion_time << "\t" << p[i].turnaround_time << "\t" <<
p[i].waiting_time << "\t" << p[i].response_time << "\t"
             << "\n"
             << endl;
    }
    cout << "Average Turnaround Time = " << avg_turnaround_time <<
endl;
    cout << "Average Waiting Time = " << avg_waiting_time << endl;
    cout << "Average Response Time = " << avg_response_time <<
endl;
}
```

SRTF

```
// Shortest Remaining Time First (SRTF)

#include <bits/stdc++.h>
using namespace std;

struct Process
{
    int pid; // Process ID
    int bt;  // Burst Time
    int art; // Arrival Time
};

// Function to find the waiting time for all
// processes
void findWaitingTime(Process proc[], int n,
                     int wt[])
{
    int rt[n];

    // Copy the burst time into rt[]
    for (int i = 0; i < n; i++)
        rt[i] = proc[i].bt;

    int complete = 0, t = 0, minm = INT_MAX;
    int shortest = 0, finish_time;
    bool check = false;

    // Process until all processes gets
    // completed
    while (complete != n)
    {

        // Find process with minimum
        // remaining time among the
        // processes that arrives till the
        // current time`
        for (int j = 0; j < n; j++)
        {
            if ((proc[j].art <= t) &&
                (rt[j] < minm) && rt[j] > 0)
            {
                minm = rt[j];
                shortest = j;
                check = true;
            }
        }

        if (check == false)
        {
            t++;
            continue;
        }

        // Reduce remaining time by one
        rt[shortest]--;

        // Update minimum
        minm = rt[shortest];
        if (minm == 0)
            minm = INT_MAX;

        // If a process gets completely
        // executed
        if (rt[shortest] == 0)
        {

            // Increment complete
            complete++;
            check = false;

            // Find finish time of current
            // process
            finish_time = t + 1;

            // Calculate waiting time
            wt[shortest] = finish_time -
```

```
                           proc[shortest].bt -
                           proc[shortest].art;

            if (wt[shortest] < 0)
                wt[shortest] = 0;
        }
        // Increment time
        t++;
    }
}

// Function to calculate turn around time
void findTurnAroundTime(Process proc[], int n,
                        int wt[], int tat[])
{
    // calculating turnaround time by adding
    // bt[i] + wt[i]
    for (int i = 0; i < n; i++)
        tat[i] = proc[i].bt + wt[i];
}

// Function to calculate average time
void findavgTime(Process proc[], int n)
{
    int wt[n], tat[n], total_wt = 0,
                       total_tat = 0;

    // Function to find waiting time of all
    // processes
    findWaitingTime(proc, n, wt);

    // Function to find turn around time for
    // all processes
    findTurnAroundTime(proc, n, wt, tat);

    // Display processes along with all
    // details
    cout << " P\t\t"
         << "BT\t\t"
         << "WT\t\t"
         << "TAT\t\t\n";

    // Calculate total waiting time and
    // total turnaround time
    for (int i = 0; i < n; i++)
    {
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        cout << " " << proc[i].pid << "\t\t"
             << proc[i].bt << "\t\t " << wt[i]
             << "\t\t " << tat[i] << endl;
    }
    cout << "\nAverage waiting time = "
         << (float)total_wt / (float)n;
    cout << "\nAverage turn around time = "
         << (float)total_tat / (float)n;
}

// Driver code
int main()
{
    int n;
    cout << "Enter the number of processes \n";
    cin >> n;
    Process proc[n];
    cout << "Enter Process ID, Burst Time and Arrival Time for "
<< n << " processes \n";
    for (int i = 0; i < n; i++)
    {
        cin >> proc[i].pid >> proc[i].bt >> proc[i].art;
    }

    findavgTime(proc, n);
    return 0;
}
```

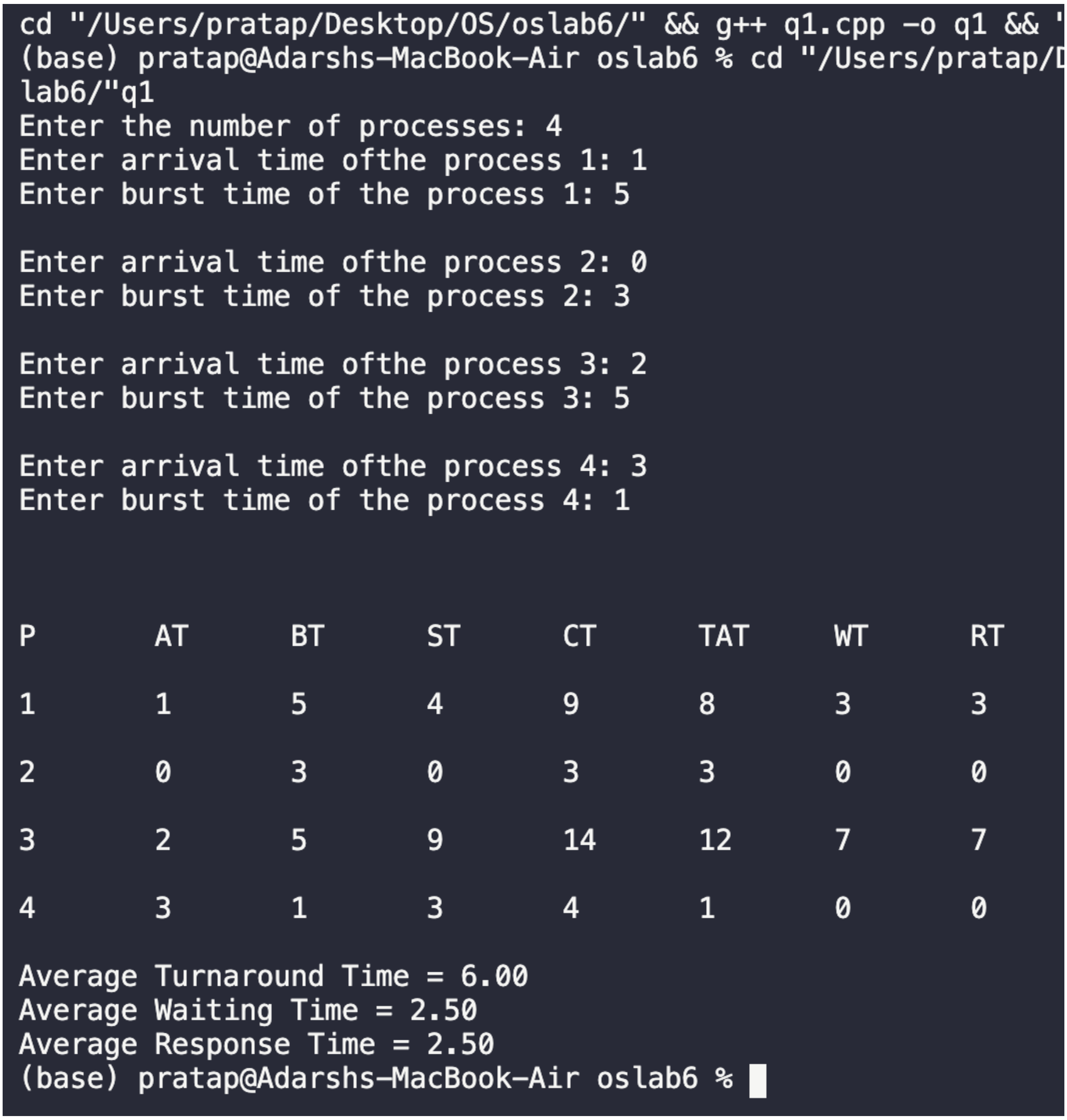
Round Robin Time Quantum 3

```
// Round Robin Algorithm with time quantum 3

#include <bits/stdc++.h>
using namespace std;

void queueUpdation(int queue[], int timer, int arrival[], int n,
int maxProccessIndex)
{
    int zeroIndex;
    for (int i = 0; i < n; i++)
    {
        if (queue[i] == 0)
        {
            zeroIndex = i;
            break;
        }
    }
    queue[zeroIndex] = maxProccessIndex + 1;
}

void queueMaintainence(int queue[], int n)
{
    for (int i = 0; (i < n - 1) && (queue[i + 1] != 0); i++)
    {
        int temp = queue[i];
        queue[i] = queue[i + 1];
        queue[i + 1] = temp;
    }
}

void checkNewArrival(int timer, int arrival[], int n, int 
maxProccessIndex, int queue[])
{
    if (timer <= arrival[n - 1])
    {
        bool newArrival = false;
        for (int j = (maxProccessIndex + 1); j < n; j++)
        {
            if (arrival[j] <= timer)
            {
                if (maxProccessIndex < j)
                {
                    maxProccessIndex = j;
                    newArrival = true;
                }
            }
        }
        // adds the incoming process to the ready queue
        //(if any arrives)
        if (newArrival)
            queueUpdation(queue, timer, arrival, n, 
maxProccessIndex);
    }
}

// Driver Code
int main()
{
    int n, tq, timer = 0, maxProccessIndex = 0;
    float avgWait = 0, avgTT = 0;
    cout << "\nEnter the time quanta : ";
    cin >> tq;
    cout << "\nEnter the number of processes : ";
    cin >> n;
    int arrival[n], burst[n], wait[n], turn[n], queue[n], 
temp_burst[n];
    bool complete[n];

    cout << "\nEnter the arrival time of the processes : ";
    for (int i = 0; i < n; i++)
        cin >> arrival[i];

    cout << "\nEnter the burst time of the processes : ";
    for (int i = 0; i < n; i++)
    {
        cin >> burst[i];
        temp_burst[i] = burst[i];
    }

    for (int i = 0; i < n; i++)
    { // Initializing the queue and complete array
        complete[i] = false;
        queue[i] = 0;
    }
    while (timer < arrival[0]) // Incrementing Timer until the
first process arrives
        timer++;
    queue[0] = 1;

    while (true)
    {
        bool flag = true;
        for (int i = 0; i < n; i++)
        {
            if (temp_burst[i] != 0)
            {
                flag = false;
                break;
            }
        }
        if (flag)
            break;

        for (int i = 0; (i < n) && (queue[i] != 0); i++)
        {
            int ctr = 0;
            while ((ctr < tq) && (temp_burst[queue[0] - 1] > 0))
            {
                temp_burst[queue[0] - 1] -= 1;
                timer += 1;
                ctr++;

                // Checking and Updating the ready queue until all
the processes arrive
                checkNewArrival(timer, arrival, n,
maxProccessIndex, queue);
            }
            // If a process is completed then store its exit time
            // and mark it as completed
            if ((temp_burst[queue[0] - 1] == 0) &&
(complete[queue[0] - 1] == false))
            {
                // turn array currently stores the completion time
                turn[queue[0] - 1] = timer;
                complete[queue[0] - 1] = true;
            }

            // checks whether or not CPU is idle
            bool idle = true;
            if (queue[n - 1] == 0)
            {
                for (int i = 0; i < n && queue[i] != 0; i++)
                {
                    if (complete[queue[i] - 1] == false)
                    {
                        idle = false;
                    }
                }
            }
            else
                idle = false;

            if (idle)
            {
                timer++;
                checkNewArrival(timer, arrival, n,
maxProccessIndex, queue);
            }

            // Maintaining the entries of processes
            // after each premption in the ready Queue
            queueMaintainence(queue, n);
        }
    }

    for (int i = 0; i < n; i++)
    {
        turn[i] = turn[i] - arrival[i];
        wait[i] = turn[i] - burst[i];
    }

    cout << "\nProgram No.\tArrival Time\tBurst Time\tWait
Time\tTurnAround Time"
         << endl;
    for (int i = 0; i < n; i++)
    {
        cout << i + 1 << "\t\t" << arrival[i] << "\t\t"
             << burst[i] << "\t\t" << wait[i] << "\t\t" << turn[i]
<< endl;
    }
    for (int i = 0; i < n; i++)
    {
        avgWait += wait[i];
        avgTT += turn[i];
    }
    cout << "\nAverage wait time : " << (avgWait / n)
         << "\nAverage Turn Around Time : " << (avgTT / n);

    return 0;
}
```

Shortest Job First

the process with the **smallest execution time** is **selected for execution next**

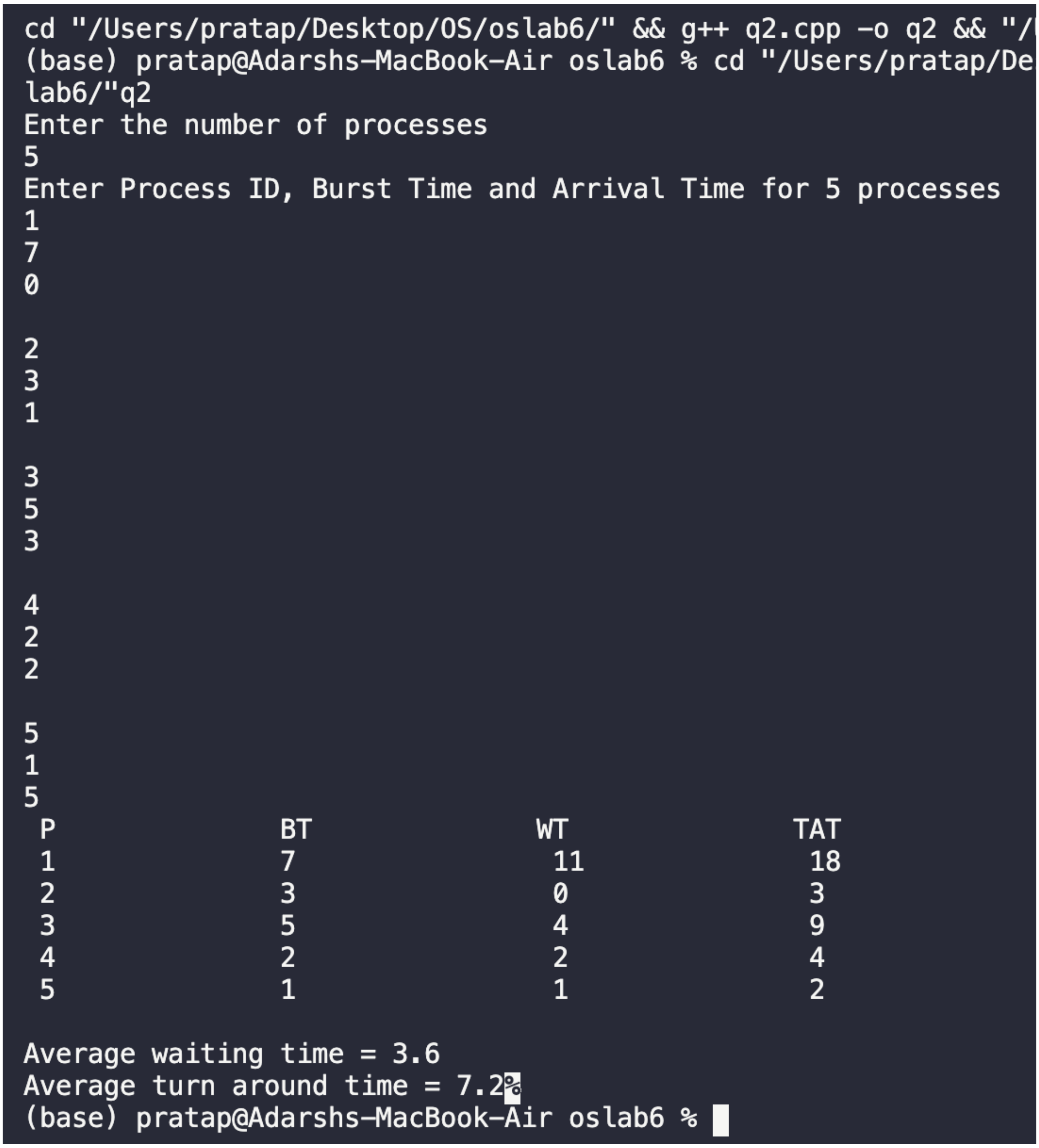
Arrival Time: the time when a process enters into the ready state and is ready for its execution
Burst Time: requires some amount of time for its execution
Completion Time: the time at which the process completes its execution
Turnaround Time: the time taken by a process since it enters a ready queue for the process of execution till the completion
Waiting Time: the total time spent by the process in the ready state waiting for CPU
Response Time: the time spent when the process is in the ready state and gets the CPU for the first time



```
cd "/Users/pratap/Desktop/OS/oslab6/" && g++ q1.cpp -o q1 && '
(base) pratap@Adarshs-MacBook-Air oslab6 % cd "/Users/pratap/D
lab6/"q1
Enter the number of processes: 4
Enter arrival time ofthe process 1: 1
Enter burst time of the process 1: 5

Enter arrival time ofthe process 2: 0
Enter burst time of the process 2: 3

Enter arrival time ofthe process 3: 2
Enter burst time of the process 3: 5

Enter arrival time ofthe process 4: 3
Enter burst time of the process 4: 1



P       AT      BT      ST      CT      TAT     WT      RT

1       1       5       4       9       8       3       3

2       0       3       0       3       3       0       0

3       2       5       9       14      12      7       7

4       3       1       3       4       1       0       0

Average Turnaround Time = 6.00
Average Waiting Time = 2.50
Average Response Time = 2.50
(base) pratap@Adarshs-MacBook-Air oslab6 % 
```

Shortest Remaining Time First (SRTF)

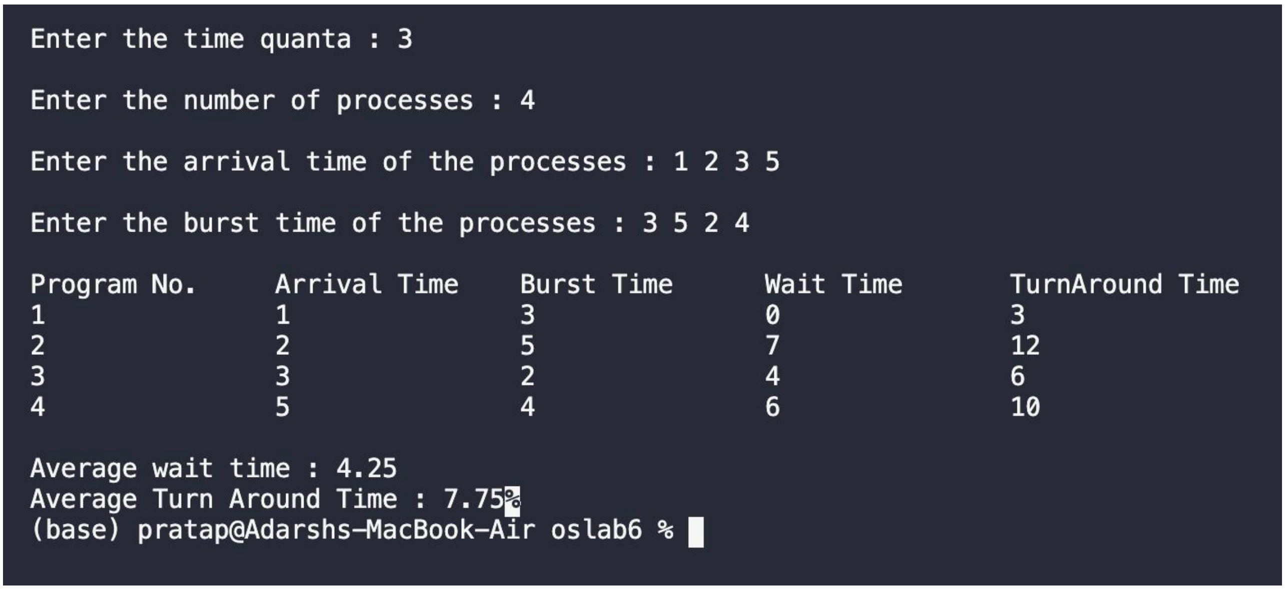
is a scheduling method that is a preemptive version of shortest job next scheduling



```
cd "/Users/pratap/Desktop/OS/oslab6/" && g++ q2.cpp -o q2 && "/
(base) pratap@Adarshs-MacBook-Air oslab6 % cd "/Users/pratap/De
lab6/"q2
Enter the number of processes
5
Enter Process ID, Burst Time and Arrival Time for 5 processes
1
7
0

2
3
1

3
5
3

4
2
2

5
1
5
 P              BT              WT              TAT
 1              7                11               18
 2              3                0                3
 3              5                4                9
 4              2                2                4
 5              1                1                2

Average waiting time = 3.6
Average turn around time = 7.2%
(base) pratap@Adarshs-MacBook-Air oslab6 %
```

Round Robin Scheduling with quantum 3 units

time slices (also known as time quanta) are assigned to each process in equal portions and in circular order, handling all processes without priority



```
Enter the time quanta : 3

Enter the number of processes : 4

Enter the arrival time of the processes : 1 2 3 5

Enter the burst time of the processes : 3 5 2 4

Program No.     Arrival Time    Burst Time      Wait Time       TurnAround Time
1               1               3               0               3
2               2               5               7               12
3               3               2               4               6
4               5               4               6               10

Average wait time : 4.25
Average Turn Around Time : 7.75%
(base) pratap@Adarshs-MacBook-Air oslab6 % 
```