Name: SHIVAM KUMAR SINGH
Reg. No.: 11803277
E-mail: shivam9608@gmail.com
Github Link: https://github.com/shivam9608/OS_SimulationProject.git
Problem Code: 16

DESCRIPTION:-

To design a scheduler that can schedule the processes arriving in the system at periodical intervals. Every process is assigned with a fixed time slice t milliseconds. If it is not able to complete its execution within the assigned time quantum, then automated timer generates an interrupt. The scheduler will select the next process in the queue and the dispatcher dispatches the process to processor for execution. The total time is computed for which processes were in the queue waiting for the processor. The input is taken for CPU burst time, arrival time and time quantum from the user.

The **Round-Robin Algorithm** (used for the scheduler) focuses on Time Sharing. In this algorithm, every process gets executed in a cyclic way. A certain time slice is defined in the system, called **time quantum**. Each process present in the ready queue is assigned the CPU for that time quantum. If the execution of the process is completed during that time, the process will terminate else the process goes back to the ready queue and waits for the next turn to complete its execution. This can be summarized as:

1. Round Robin is the pre-emptive process scheduling algorithm.
2. Each process is provided a fix time to execute, it is called a time-quantum.
3. Once a process is executed for a given time period, it is pre-empted and other process executes for a given time period.
4. Context switching is used to save states of pre-empted processes.

ALGORITHM:-

1. Create an array arrival_time, burst_time to keep track of arrival and bust time of processes.
2. Create another array temp to store waiting times of processes temporarily in between execution.
3. Initialize time: total = 0, counter = 0, wait time = 0, turnaround time = 0.
4. Ask user for no of process and store it in limit.
5. Repetitively ask user to give input for - arrival time, bust time up to limit.
6. Ask user to enter time quantum and store it into time_quantum.
7. Keep traversing the all processes while all processes are not done. Do following for i'th process if it is not done yet.

Name: SHIVAM KUMAR SINGH
Reg. No.: 11803277
E-mail: shivam9608@gmail.com
Github Link: https://github.com/shivam9608/OS_SimulationProject.git
Problem Code: 16

```
a. If temp[i] <= time_quantum && temp[i] > 0
        (i) total = total + temp[i];
        (ii) temp[i] = 0;
        (iii) counter = 1;
b. Else if temp[i] > 0
        (i) temp[i] = temp[i] - time_quantum;
        (ii) total = total + time_quantum;
c. If temp[i] == 0 && counter == 1
        (i) x--;
        (ii) print burst_time[i], total, arrival_time[i],
             total - arrival_time[i] -burst_time[i]);
        (iii) wait_time = wait_time + total - arrival_time[i] -
    burst_time[i];
        (iv) turnaround_time = turnaround_time + total - arrival_time[i];
        (v) counter = 0;
    d. If i == limit – 1 (i) i = 0;
    e. Else if arrival_time[i + 1] <= total (i) i++; c. Else (i) i = 0;
```

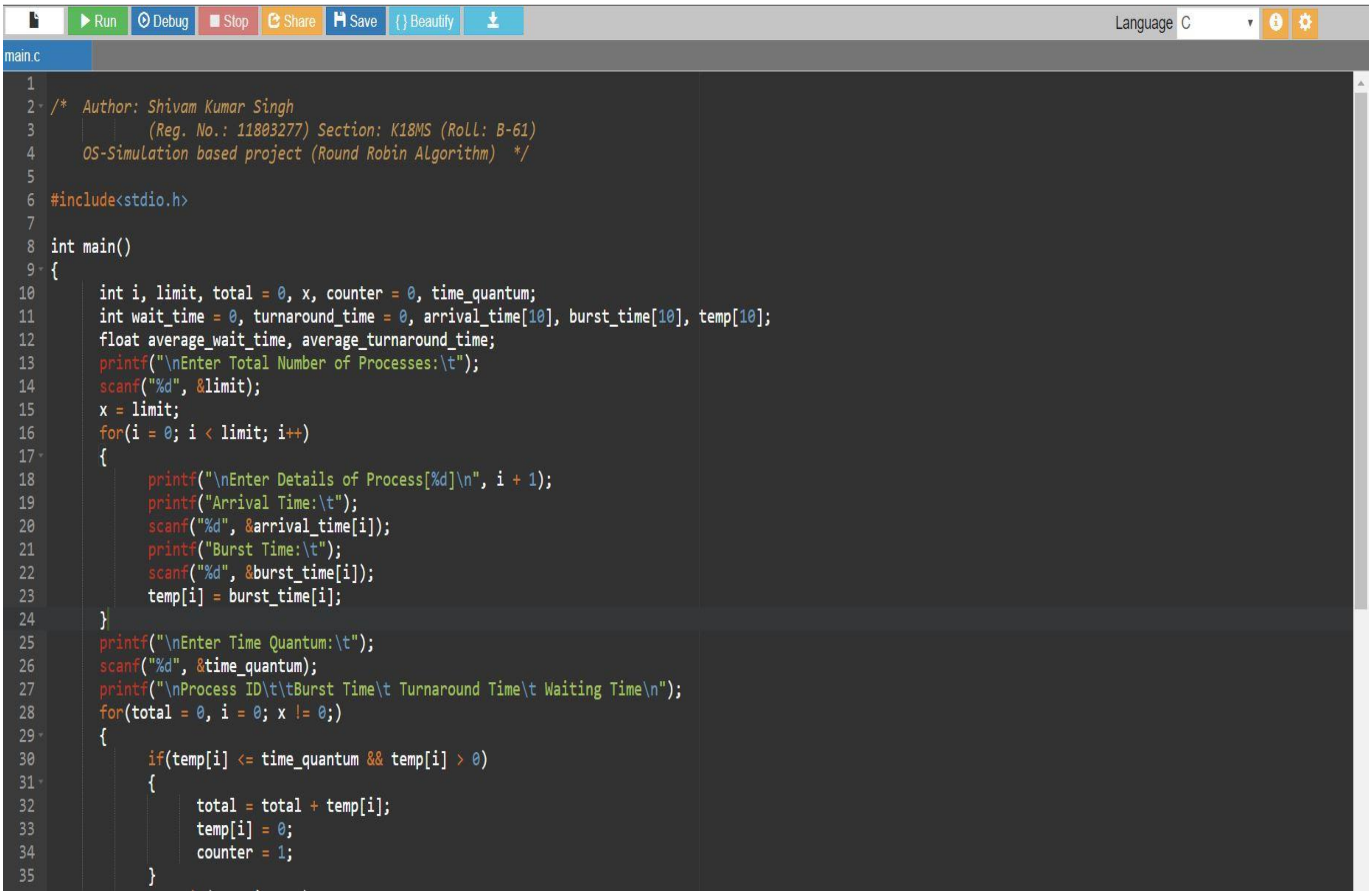
8. Print total waiting time - wait_time, average waiting time - wait_time/limit, average turnaround time - average_turnaround_time/limit.

Name: SHIVAM KUMAR SINGH
Reg. No.: 11803277
E-mail: shivam9608@gmail.com
Github Link: https://github.com/shivam9608/OS_SimulationProject.git
Problem Code: 16

COMPLEXITY:-

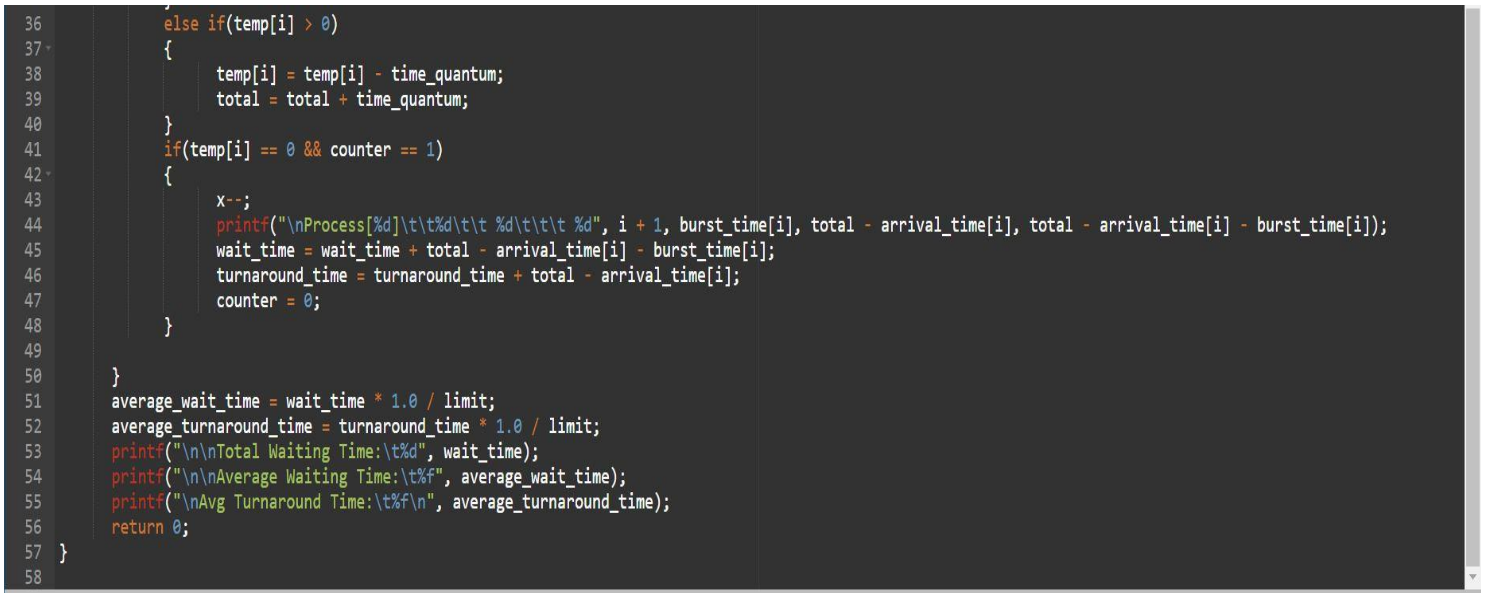
- Complexity of initialization of variables is O(1). total = 0 counter = 0 wait_time = 0 turnaround_time = 0
- Complexity for initializing arrays with for loop is O(limit). for(i = 0; i < limit; i++) { … }
- Complexity of calculating waiting and turnaround time is O(limit). for(total = 0, i = 0; x != 0;) { …. }
- Complexity of implemented algorithm is O(limit).

CODE SNIPPETS:-



```c

/*  Author: Shivam Kumar Singh
            (Reg. No.: 11803277) Section: K18MS (Roll: B-61)
    OS-Simulation based project (Round Robin Algorithm)  */

#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;
            }
```

Name: SHIVAM KUMAR SINGH
Reg. No.: 11803277
E-mail: shivam9608@gmail.com
Github Link: https://github.com/shivam9608/OS_SimulationProject.git
Problem Code: 16



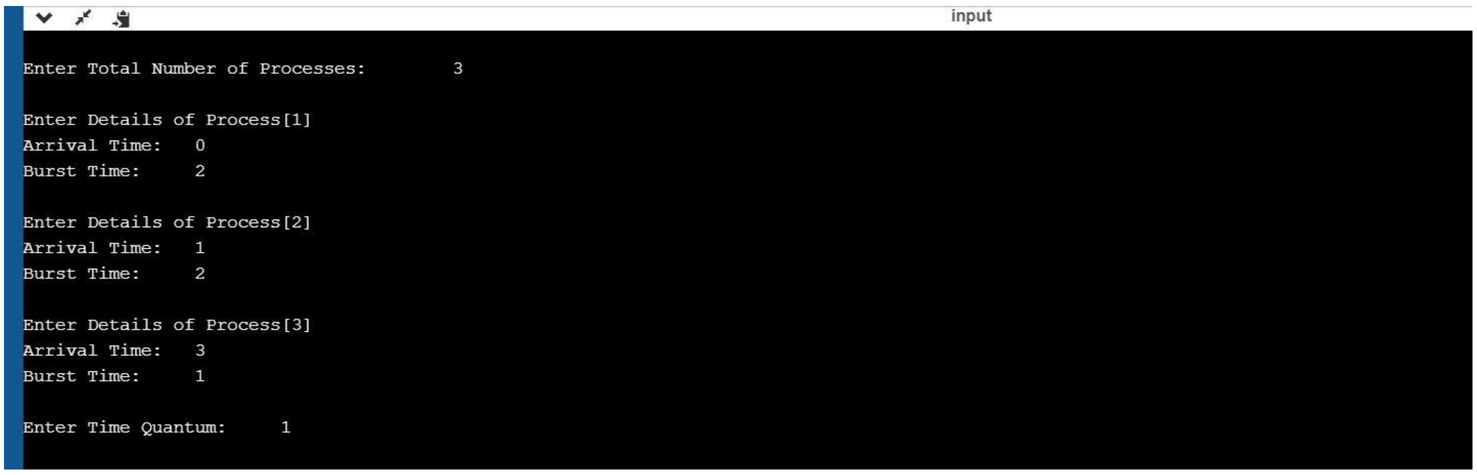
```c
            else if(temp[i] > 0)
            {
                  temp[i] = temp[i] - time_quantum;
                  total = total + time_quantum;
            }
            if(temp[i] == 0 && counter == 1)
            {
                  x--;
                  printf("\nProcess[%d]\t\t%d\t\t %d\t\t\t %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;
            }

      }
      average_wait_time = wait_time * 1.0 / limit;
      average_turnaround_time = turnaround_time * 1.0 / limit;
      printf("\n\nTotal Waiting Time:\t%d", wait_time);
      printf("\n\nAverage Waiting Time:\t%f", average_wait_time);
      printf("\nAvg Turnaround Time:\t%f\n", average_turnaround_time);
      return 0;
}
```

CONSTRAINTS:-

- If we have large number of processes and a process with very less burst time. According to round robin it will have to wait for very long time if burst time of other processes is large and its turn comes late.

- If we have some processes running according to round robin and we have a process whose arrival time is after the completion of those processes, then that process will not execute. If we want that all the processes should execute successfully then the arrival time of other process must be less than or equal to the completion time of the running processes.

TEST CASES:-



```
Enter Total Number of Processes:        3

Enter Details of Process[1]
Arrival Time:   0
Burst Time:     2

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

Enter Details of Process[3]
Arrival Time:   3
Burst Time:     1

Enter Time Quantum:     1
```

Name: SHIVAM KUMAR SINGH
Reg. No.: 11803277
E-mail: shivam9608@gmail.com
Github Link: https://github.com/shivam9608/OS_SimulationProject.git
Problem Code: 16

1. If we select same arrival time for 2 processes. Expected Result: The process having less burst time should have less turnaround time.

   Test case passed successfully.

2. If p1 process arrives at 2 and burst time=4 but p2 and p3 process arrives at 6. Expected Result: Here only process p1 will execute because p2 and p3 are unable to arrive within the completion of process p1.

   Test case passed successfully.

3. Now if process p1 arrives at 0 and two other processes p2 and p3 arrives within the completion of process p1. Expected Result: All 3 process will execute.

   Test case passed successfully.

Git-Hub Link:-

https://github.com/shivam9608/OS_SimulationProject.git