Ex. No. 6 a

FIRST COME FIRST SERVE

Aim:

To implement First-come First- serve (FCFS) scheduling technique

Algorithm:

1. Get the number of processes from the user.

2. Read the process name and burst time.

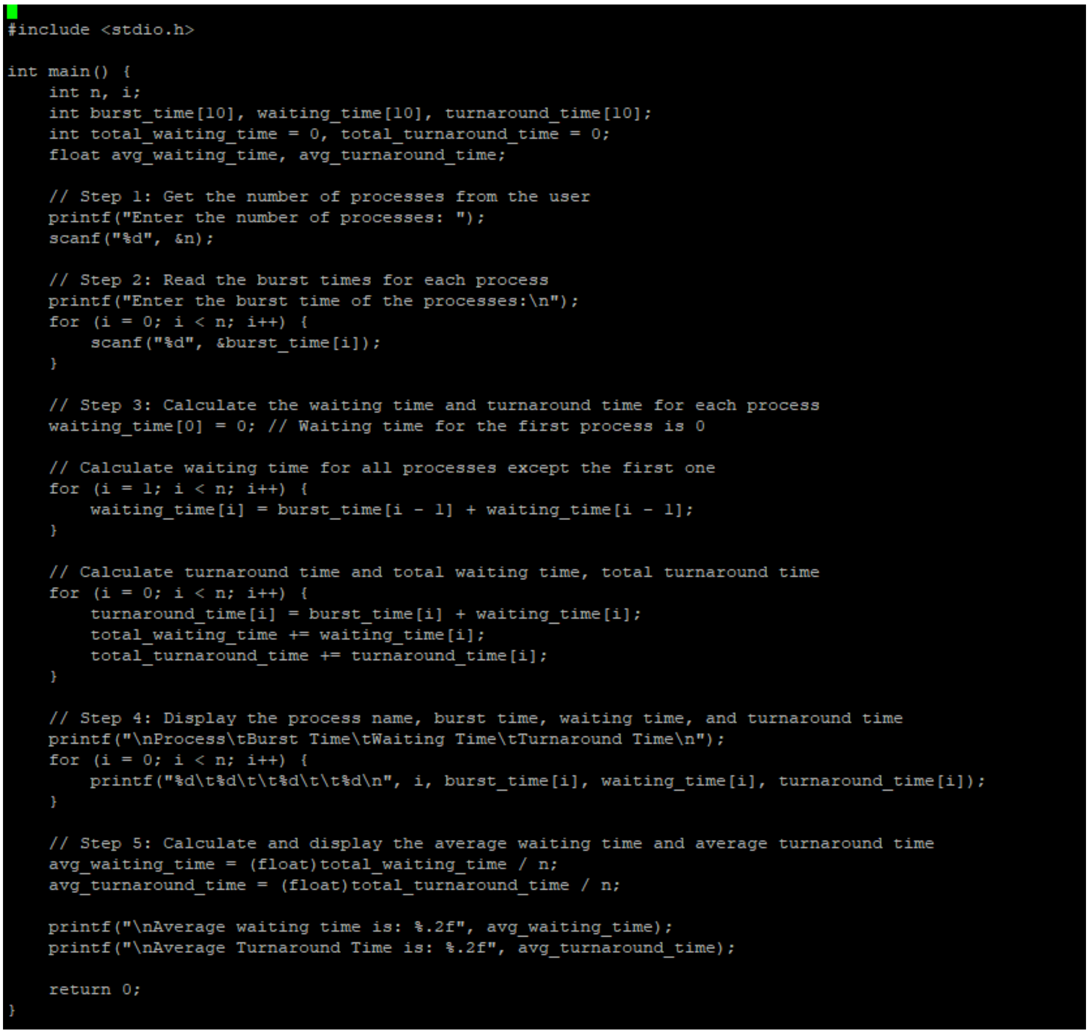
3. Calculate the total process time.

4. Calculate the total waiting time and total turnaround time for each process 5.

Display the process name & burst time for each process. 6. Display the total

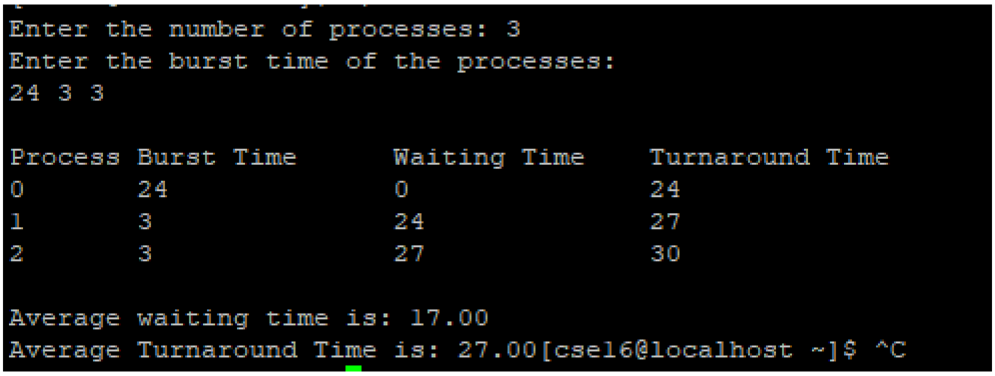
waiting time, average waiting time, turnaround time

Program Code:



```
#include <stdio.h>

int main() {
    int n, i;
    int burst_time[10], waiting_time[10], turnaround_time[10];
    int total_waiting_time = 0, total_turnaround_time = 0;
    float avg_waiting_time, avg_turnaround_time;

    // Step 1: Get the number of processes from the user
    printf("Enter the number of processes: ");
    scanf("%d", &n);

    // Step 2: Read the burst times for each process
    printf("Enter the burst time of the processes:\n");
    for (i = 0; i < n; i++) {
        scanf("%d", &burst_time[i]);
    }

    // Step 3: Calculate the waiting time and turnaround time for each process
    waiting_time[0] = 0; // Waiting time for the first process is 0

    // Calculate waiting time for all processes except the first one
    for (i = 1; i < n; i++) {
        waiting_time[i] = burst_time[i - 1] + waiting_time[i - 1];
    }

    // Calculate turnaround time and total waiting time, total turnaround time
    for (i = 0; i < n; i++) {
        turnaround_time[i] = burst_time[i] + waiting_time[i];
        total_waiting_time += waiting_time[i];
        total_turnaround_time += turnaround_time[i];
    }

    // Step 4: Display the process name, burst time, waiting time, and turnaround time
    printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for (i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\n", i, burst_time[i], waiting_time[i], turnaround_time[i]);
    }

    // Step 5: Calculate and display the average waiting time and average turnaround time
    avg_waiting_time = (float)total_waiting_time / n;
    avg_turnaround_time = (float)total_turnaround_time / n;

    printf("\nAverage waiting time is: %.2f", avg_waiting_time);
    printf("\nAverage Turnaround Time is: %.2f", avg_turnaround_time);

    return 0;
}
```

OUTPUT-



```
Enter the number of processes: 3
Enter the burst time of the processes:
24 3 3

Process Burst Time      Waiting Time    Turnaround Time
0       24              0               24
1       3               24              27
2       3               27              30

Average waiting time is: 17.00
Average Turnaround Time is: 27.00[cse16@localhost ~]$ ^C
```

Ex. No. 6 b

SHORTEST JOB FIRST

Aim:

To implement the Shortest Job First (SJF) scheduling technique

Algorithm:

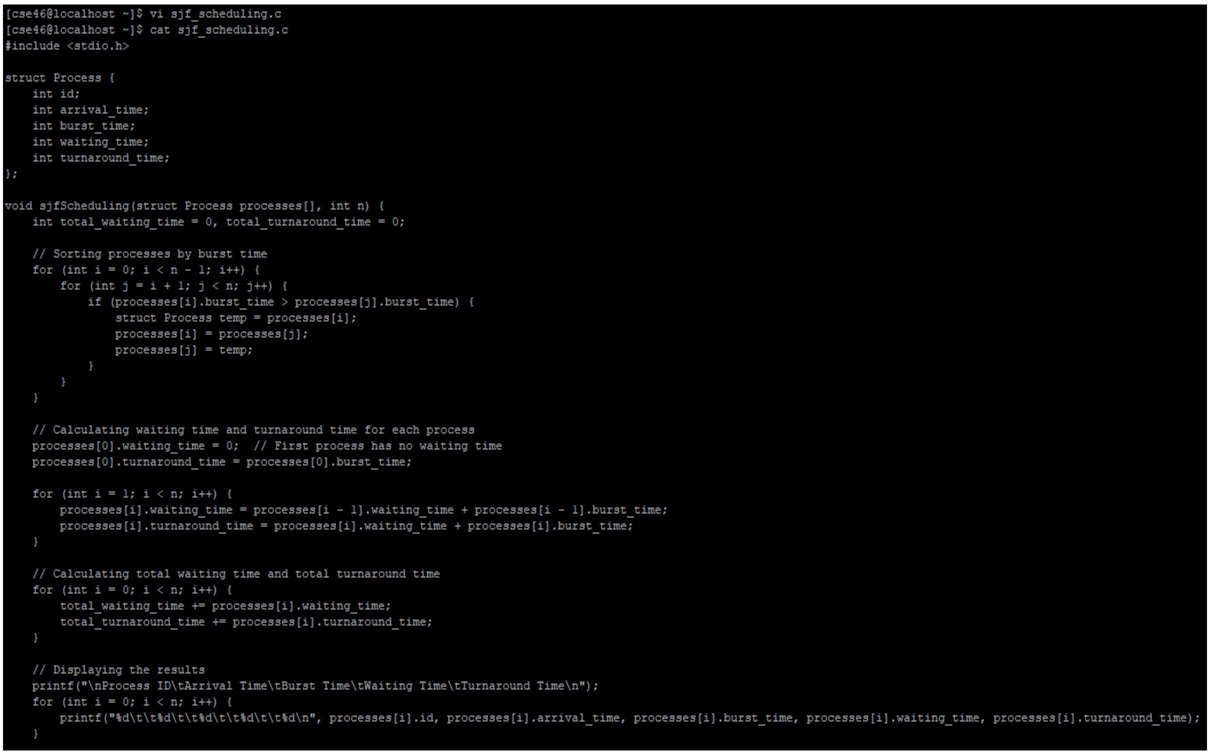
1. Declare the structure and its elements.

2. Get number of processes as input from the user.

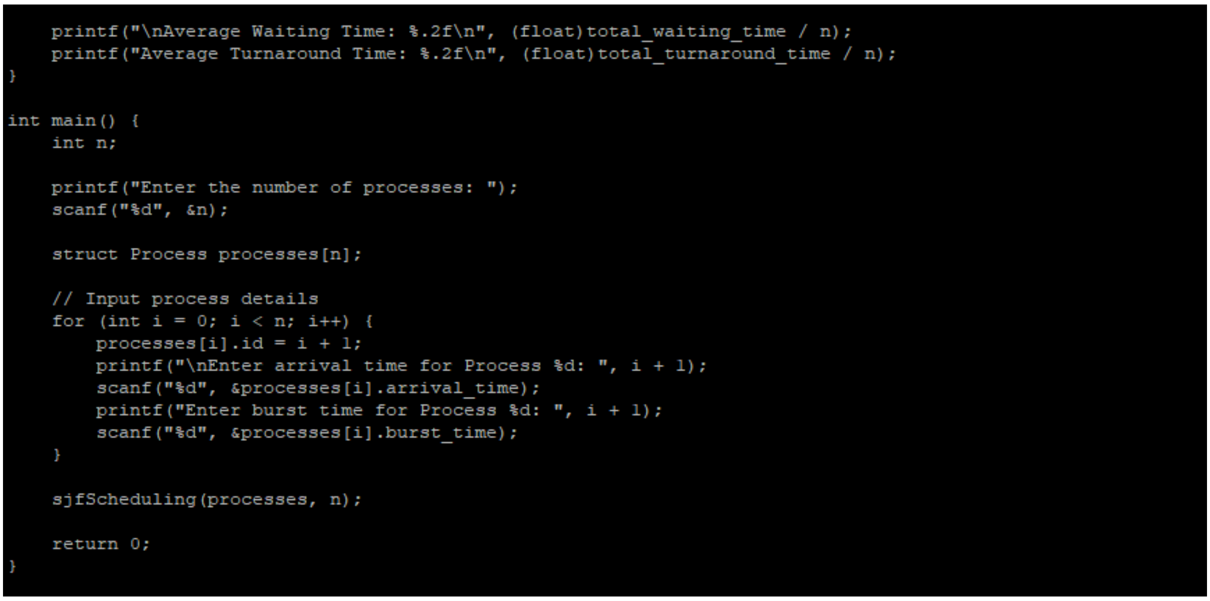
3. Read the process name, arrival time and burst time

4. Initialize waiting time, turnaround time & flag of read processes to zero. 5. Sort based on burst time of all processes in ascending order 6. Calculate the waiting time and turnaround time for each process. 7. Calculate the average waiting time and average turnaround time. 8. Display the results.

Program Code:

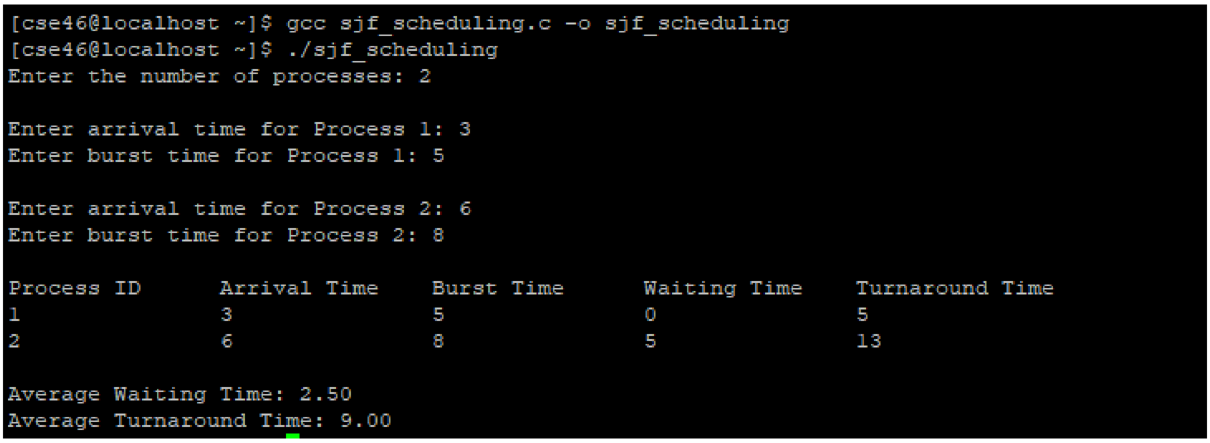


```
[cse46@localhost ~]$ vi sjf_scheduling.c
[cse46@localhost ~]$ cat sjf_scheduling.c
#include <stdio.h>

struct Process {
    int id;
    int arrival_time;
    int burst_time;
    int waiting_time;
    int turnaround_time;
};

void sjfScheduling(struct Process processes[], int n) {
    int total_waiting_time = 0, total_turnaround_time = 0;

    // Sorting processes by burst time
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (processes[i].burst_time > processes[j].burst_time) {
                struct Process temp = processes[i];
                processes[i] = processes[j];
                processes[j] = temp;
            }
        }
    }

    // Calculating waiting time and turnaround time for each process
    processes[0].waiting_time = 0;  // First process has no waiting time
    processes[0].turnaround_time = processes[0].burst_time;

    for (int i = 1; i < n; i++) {
        processes[i].waiting_time = processes[i - 1].waiting_time + processes[i - 1].burst_time;
        processes[i].turnaround_time = processes[i].waiting_time + processes[i].burst_time;
    }

    // Calculating total waiting time and total turnaround time
    for (int i = 0; i < n; i++) {
        total_waiting_time += processes[i].waiting_time;
        total_turnaround_time += processes[i].turnaround_time;
    }

    // Displaying the results
    printf("\nProcess ID\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n", processes[i].id, processes[i].arrival_time, processes[i].burst_time, processes[i].waiting_time, processes[i].turnaround_time);
    }
```



```
    printf("\nAverage Waiting Time: %.2f\n", (float)total_waiting_time / n);
    printf("Average Turnaround Time: %.2f\n", (float)total_turnaround_time / n);
}

int main() {
    int n;

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

    struct Process processes[n];

    // Input process details
    for (int i = 0; i < n; i++) {
        processes[i].id = i + 1;
        printf("\nEnter arrival time for Process %d: ", i + 1);
        scanf("%d", &processes[i].arrival_time);
        printf("Enter burst time for Process %d: ", i + 1);
        scanf("%d", &processes[i].burst_time);
    }

    sjfScheduling(processes, n);

    return 0;
}
```

Output:



```
[cse46@localhost ~]$ gcc sjf_scheduling.c -o sjf_scheduling
[cse46@localhost ~]$ ./sjf_scheduling
Enter the number of processes: 2

Enter arrival time for Process 1: 3
Enter burst time for Process 1: 5

Enter arrival time for Process 2: 6
Enter burst time for Process 2: 8

Process ID      Arrival Time    Burst Time      Waiting Time    Turnaround Time
1               3               5               0               5
2               6               8               5               13

Average Waiting Time: 2.50
Average Turnaround Time: 9.00
```

Ex. No. 6 c

PRIORITY SCHEDULING

Aim:

To implement priority scheduling technique

Algorithm:

1. Get the number of processes from the user.

2. Read the process name, burst time and priority of process.

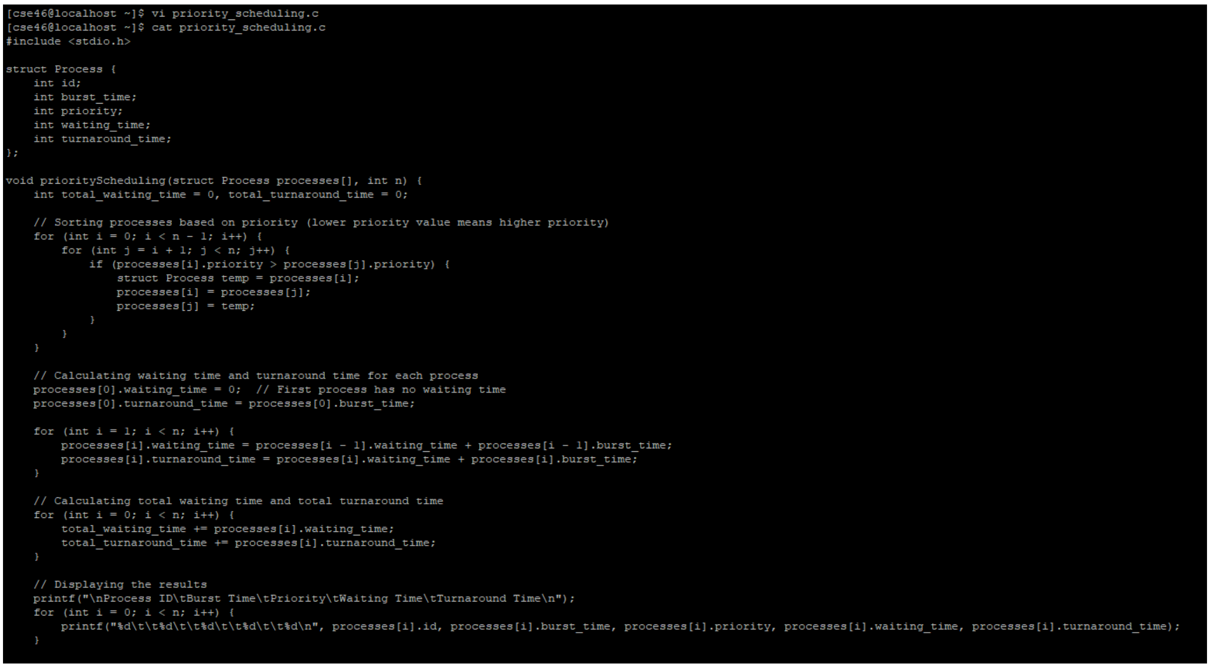
3. Sort based on burst time of all processes in ascending order based priority 4.

Calculate the total waiting time and total turnaround time for each process 5.

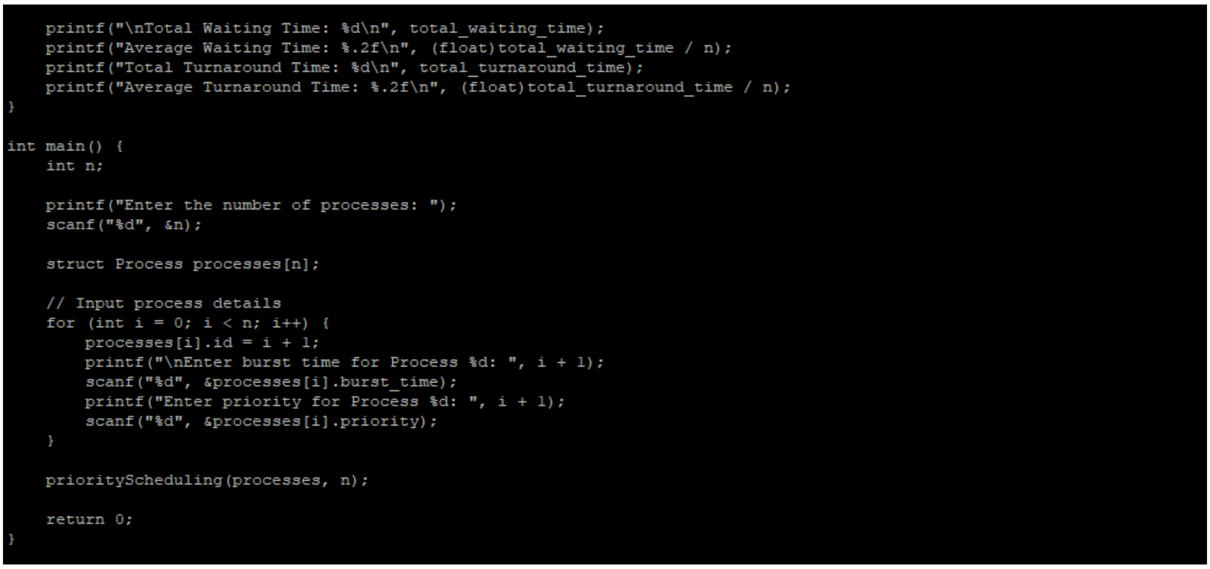
Display the process name & burst time for each process.

6. Display the total waiting time, average waiting time, turnaround time

Program Code:

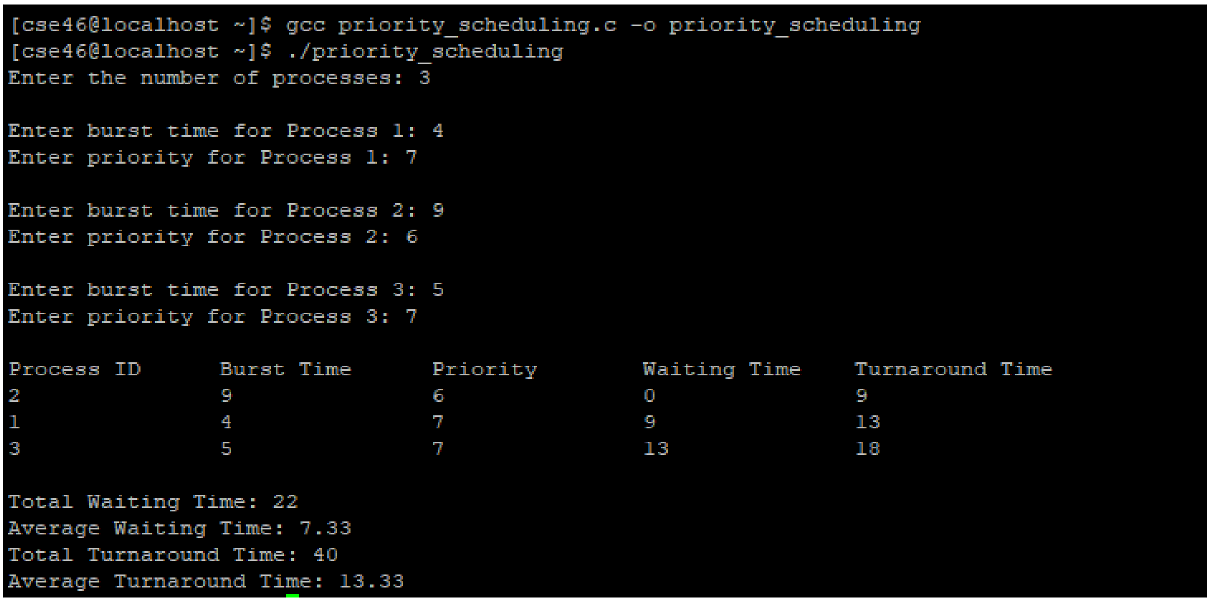


```
[cse46@localhost ~]$ vi priority_scheduling.c
[cse46@localhost ~]$ cat priority_scheduling.c
#include <stdio.h>

struct Process {
    int id;
    int burst_time;
    int priority;
    int waiting_time;
    int turnaround_time;
};

void priorityScheduling(struct Process processes[], int n) {
    int total_waiting_time = 0, total_turnaround_time = 0;

    // Sorting processes based on priority (lower priority value means higher priority)
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (processes[i].priority > processes[j].priority) {
                struct Process temp = processes[i];
                processes[i] = processes[j];
                processes[j] = temp;
            }
        }
    }

    // Calculating waiting time and turnaround time for each process
    processes[0].waiting_time = 0;  // First process has no waiting time
    processes[0].turnaround_time = processes[0].burst_time;

    for (int i = 1; i < n; i++) {
        processes[i].waiting_time = processes[i - 1].waiting_time + processes[i - 1].burst_time;
        processes[i].turnaround_time = processes[i].waiting_time + processes[i].burst_time;
    }

    // Calculating total waiting time and total turnaround time
    for (int i = 0; i < n; i++) {
        total_waiting_time += processes[i].waiting_time;
        total_turnaround_time += processes[i].turnaround_time;
    }

    // Displaying the results
    printf("\nProcess ID\tBurst Time\tPriority\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n", processes[i].id, processes[i].burst_time, processes[i].priority, processes[i].waiting_time, processes[i].turnaround_time);
    }
```



```
    printf("\nTotal Waiting Time: %d\n", total_waiting_time);
    printf("Average Waiting Time: %.2f\n", (float)total_waiting_time / n);
    printf("Total Turnaround Time: %d\n", total_turnaround_time);
    printf("Average Turnaround Time: %.2f\n", (float)total_turnaround_time / n);
}

int main() {
    int n;

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

    struct Process processes[n];

    // Input process details
    for (int i = 0; i < n; i++) {
        processes[i].id = i + 1;
        printf("\nEnter burst time for Process %d: ", i + 1);
        scanf("%d", &processes[i].burst_time);
        printf("Enter priority for Process %d: ", i + 1);
        scanf("%d", &processes[i].priority);
    }

    priorityScheduling(processes, n);

    return 0;
}
```

Output:



```
[cse46@localhost ~]$ gcc priority_scheduling.c -o priority_scheduling
[cse46@localhost ~]$ ./priority_scheduling
Enter the number of processes: 3

Enter burst time for Process 1: 4
Enter priority for Process 1: 7

Enter burst time for Process 2: 9
Enter priority for Process 2: 6

Enter burst time for Process 3: 5
Enter priority for Process 3: 7

Process ID      Burst Time      Priority        Waiting Time    Turnaround Time
2               9               6               0               9
1               4               7               9               13
3               5               7               13              18

Total Waiting Time: 22
Average Waiting Time: 7.33
Total Turnaround Time: 40
Average Turnaround Time: 13.33
```

Ex. No. 6 d

ROUND ROBIN SCHEDULING

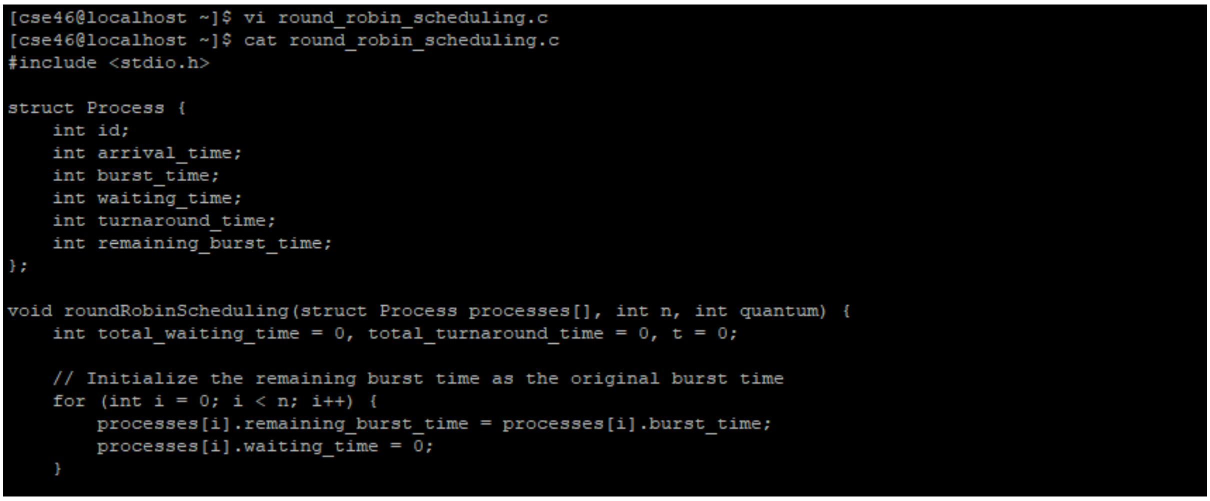
Aim:

To implement the Round Robin (RR) scheduling technique

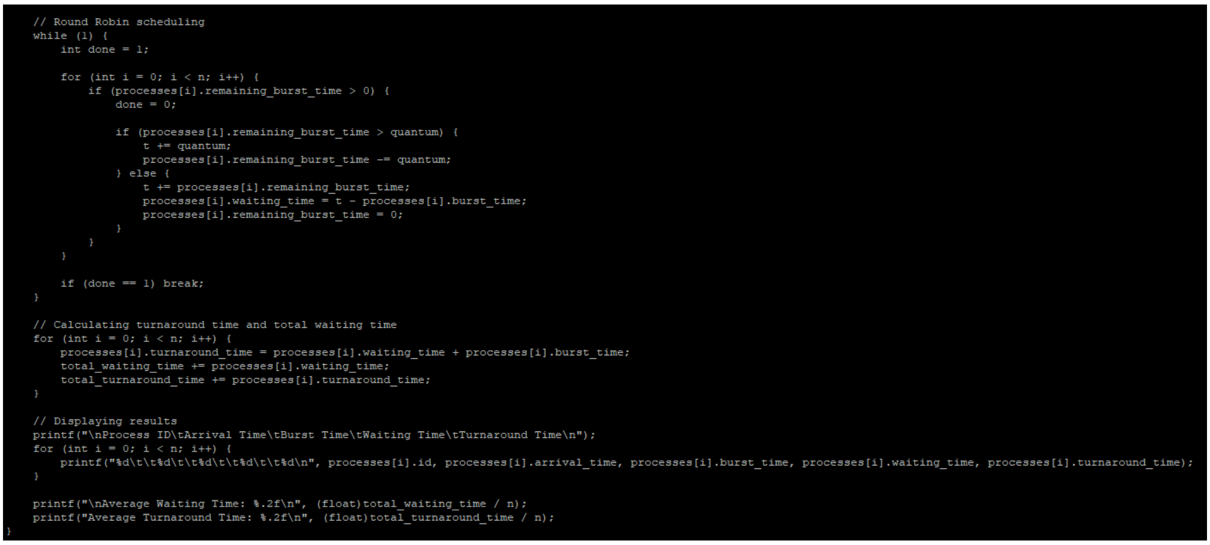
Algorithm:

```
1. Declare the structure and its elements.
2. Get number of processes and Time quantum as input from the user.
3. Read the process name, arrival time and burst time
4. Create an array rem_bt[] to keep track of remaining burst time of processes which is initially copy of bt[] (burst times array)
5. Create another array wt[] to store waiting times of processes. Initialize this array as 0. 6. Initialize time : t = 0
7. Keep traversing the all processes while all processes are not done. Do following for i'th process if it is not done yet.
a- If rem_bt[i] > quantum
(i) t = t + quantum
(ii) bt_rem[i] -= quantum;
b- Else // Last cycle for this process
(i) t = t + bt_rem[i];
(ii) wt[i] = t - bt[i]
(iii) bt_rem[i] = 0; // This process is over
8. Calculate the waiting time and turnaround time for each process.
9. Calculate the average waiting time and average turnaround time.
10. Display the results.
```

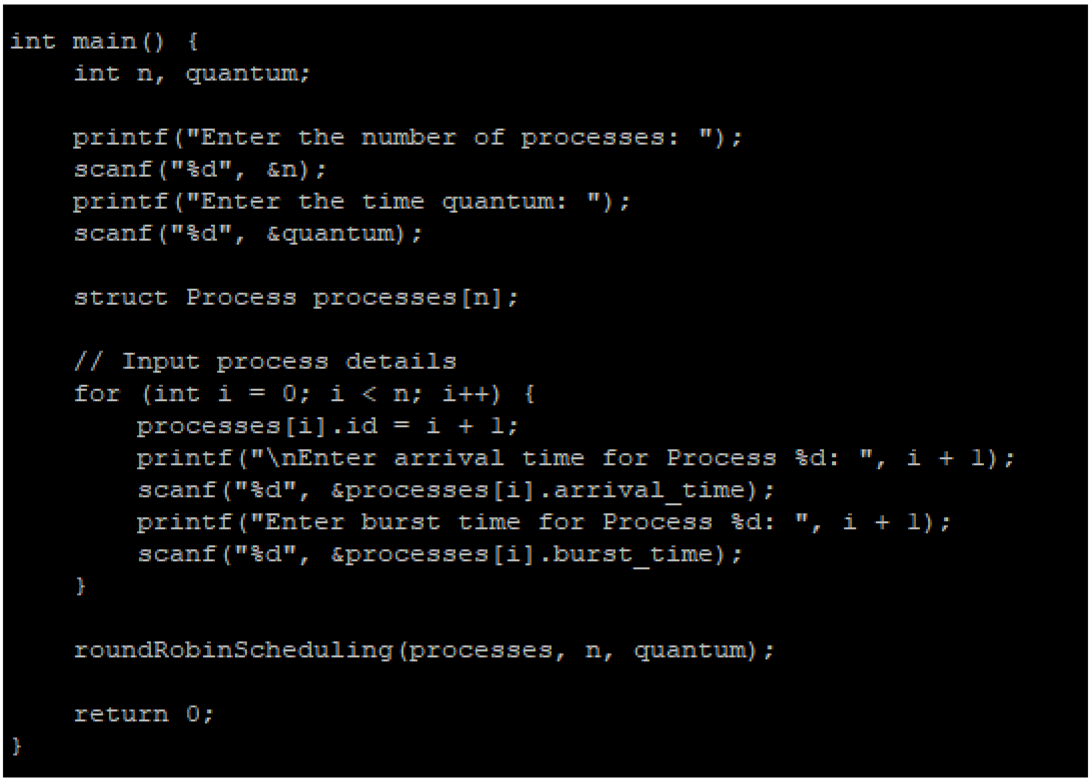
Program Code:



```
[cse46@localhost ~]$ vi round_robin_scheduling.c
[cse46@localhost ~]$ cat round_robin_scheduling.c
#include <stdio.h>

struct Process {
    int id;
    int arrival_time;
    int burst_time;
    int waiting_time;
    int turnaround_time;
    int remaining_burst_time;
};

void roundRobinScheduling(struct Process processes[], int n, int quantum) {
    int total_waiting_time = 0, total_turnaround_time = 0, t = 0;

    // Initialize the remaining burst time as the original burst time
    for (int i = 0; i < n; i++) {
        processes[i].remaining_burst_time = processes[i].burst_time;
        processes[i].waiting_time = 0;
    }
```

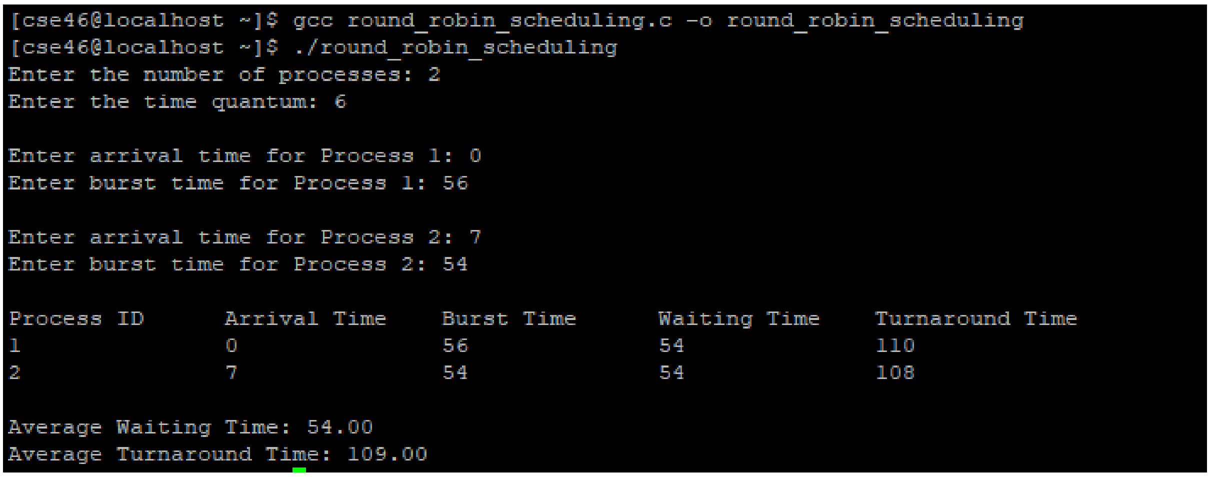


```
    // Round Robin scheduling
    while (1) {
        int done = 1;

        for (int i = 0; i < n; i++) {
            if (processes[i].remaining_burst_time > 0) {
                done = 0;

                if (processes[i].remaining_burst_time > quantum) {
                    t += quantum;
                    processes[i].remaining_burst_time -= quantum;
                } else {
                    t += processes[i].remaining_burst_time;
                    processes[i].waiting_time = t - processes[i].burst_time;
                    processes[i].remaining_burst_time = 0;
                }
            }
        }

        if (done == 1) break;
    }

    // Calculating turnaround time and total waiting time
    for (int i = 0; i < n; i++) {
        processes[i].turnaround_time = processes[i].waiting_time + processes[i].burst_time;
        total_waiting_time += processes[i].waiting_time;
        total_turnaround_time += processes[i].turnaround_time;
    }

    // Displaying results
    printf("\nProcess ID\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n", processes[i].id, processes[i].arrival_time, processes[i].burst_time, processes[i].waiting_time, processes[i].turnaround_time);
    }

    printf("\nAverage Waiting Time: %.2f\n", (float)total_waiting_time / n);
    printf("Average Turnaround Time: %.2f\n", (float)total_turnaround_time / n);
}
```



```
int main() {
    int n, quantum;

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

    struct Process processes[n];

    // Input process details
    for (int i = 0; i < n; i++) {
        processes[i].id = i + 1;
        printf("\nEnter arrival time for Process %d: ", i + 1);
        scanf("%d", &processes[i].arrival_time);
        printf("Enter burst time for Process %d: ", i + 1);
        scanf("%d", &processes[i].burst_time);
    }

    roundRobinScheduling(processes, n, quantum);

    return 0;
}
```

Output:



```
[cse46@localhost ~]$ gcc round_robin_scheduling.c -o round_robin_scheduling
[cse46@localhost ~]$ ./round_robin_scheduling
Enter the number of processes: 2
Enter the time quantum: 6

Enter arrival time for Process 1: 0
Enter burst time for Process 1: 56

Enter arrival time for Process 2: 7
Enter burst time for Process 2: 54

Process ID      Arrival Time    Burst Time      Waiting Time    Turnaround Time
1               0               56              54              110
2               7               54              54              108

Average Waiting Time: 54.00
Average Turnaround Time: 109.00
```