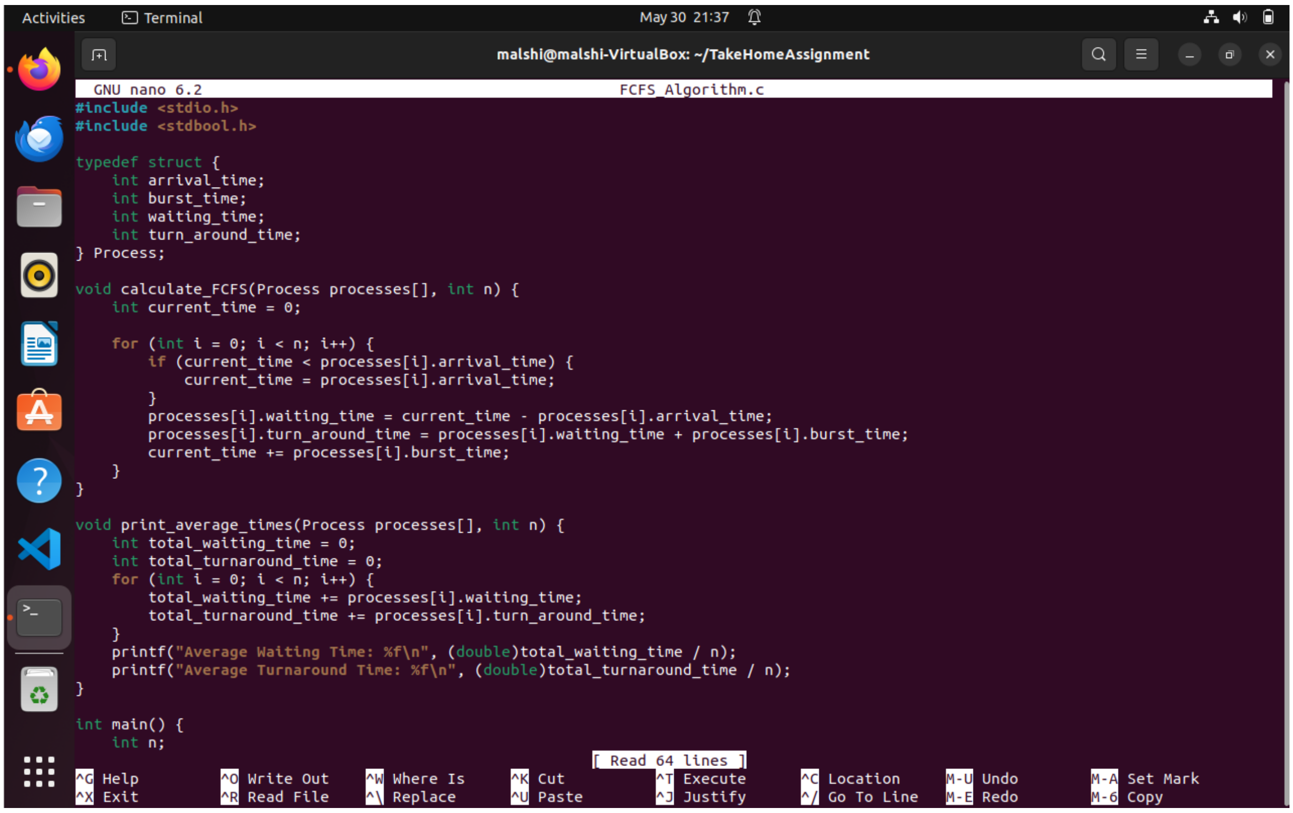
EE 6253 : OPERATING SYSTEMS AND NETWORK PROGRAMMING

TAKE HOME ASSIGNMENT

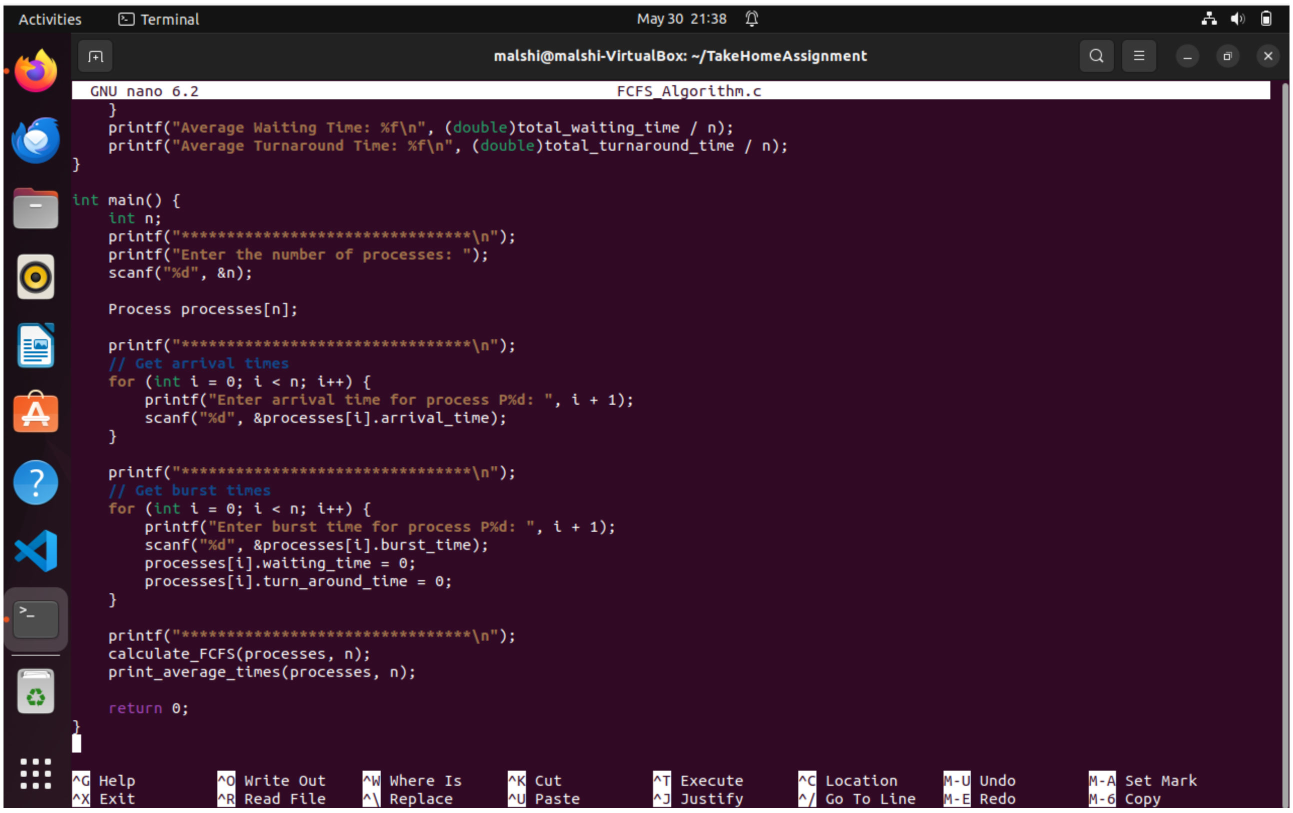
NAME	: GIMNADHI P.M.T.
REG NO.	: EG/2020/3942
SEMESTER	: 06
DATE	: 01/05/2024

Source codes of implementation

- **FCFS Scheduling Algorithm**

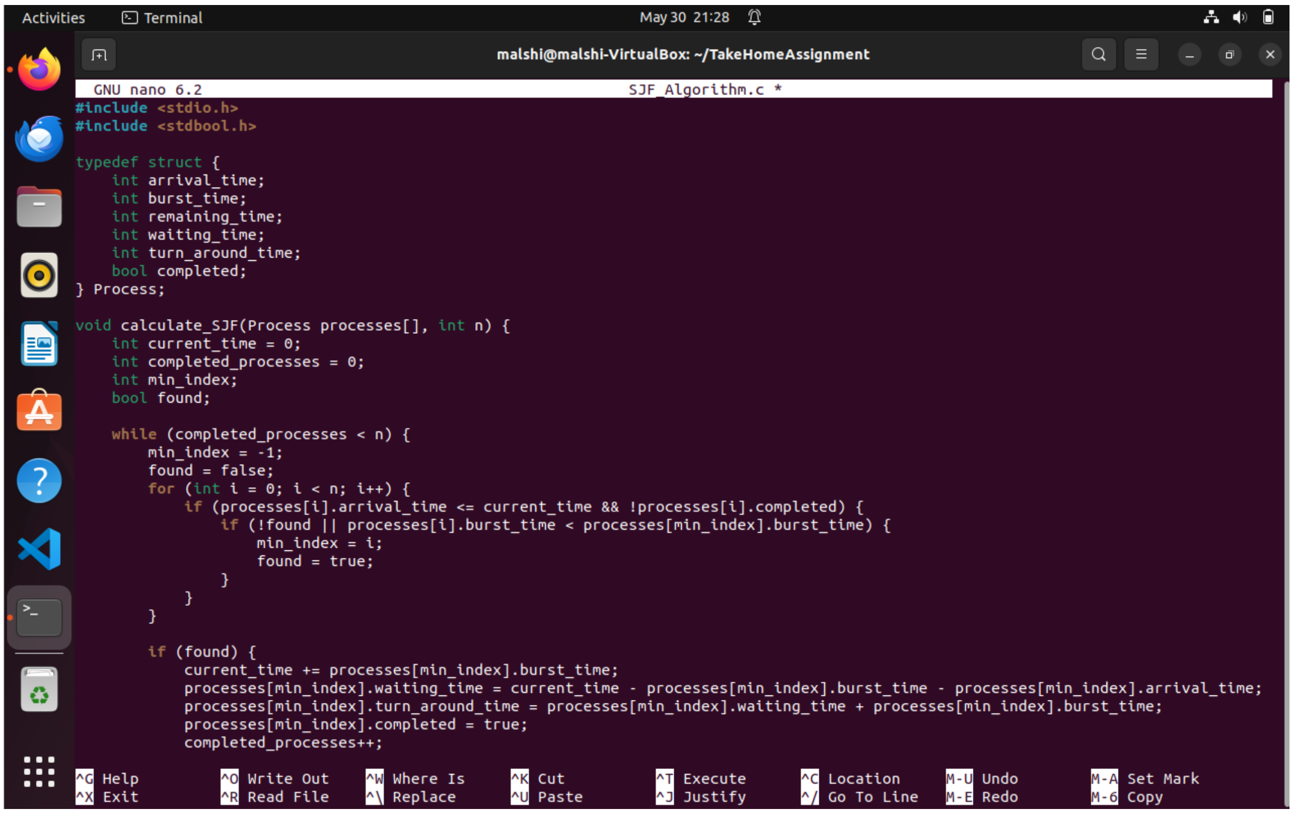


```
  GNU nano 6.2                                       FCFS_Algorithm.c
#include <stdio.h>
#include <stdbool.h>

typedef struct {
    int arrival_time;
    int burst_time;
    int waiting_time;
    int turn_around_time;
} Process;

void calculate_FCFS(Process processes[], int n) {
    int current_time = 0;

    for (int i = 0; i < n; i++) {
        if (current_time < processes[i].arrival_time) {
            current_time = processes[i].arrival_time;
        }
        processes[i].waiting_time = current_time - processes[i].arrival_time;
        processes[i].turn_around_time = processes[i].waiting_time + processes[i].burst_time;
        current_time += processes[i].burst_time;
    }
}

void print_average_times(Process processes[], int n) {
    int total_waiting_time = 0;
    int total_turnaround_time = 0;
    for (int i = 0; i < n; i++) {
        total_waiting_time += processes[i].waiting_time;
        total_turnaround_time += processes[i].turn_around_time;
    }
    printf("Average Waiting Time: %f\n", (double)total_waiting_time / n);
    printf("Average Turnaround Time: %f\n", (double)total_turnaround_time / n);
}

int main() {
    int n;
                                               [ Read 64 lines ]
^G Help         ^O Write Out    ^W Where Is     ^K Cut          ^T Execute      ^C Location     M-U Undo        M-A Set Mark
^X Exit         ^R Read File    ^\ Replace      ^U Paste        ^J Justify      ^/ Go To Line   M-E Redo        M-6 Copy
```

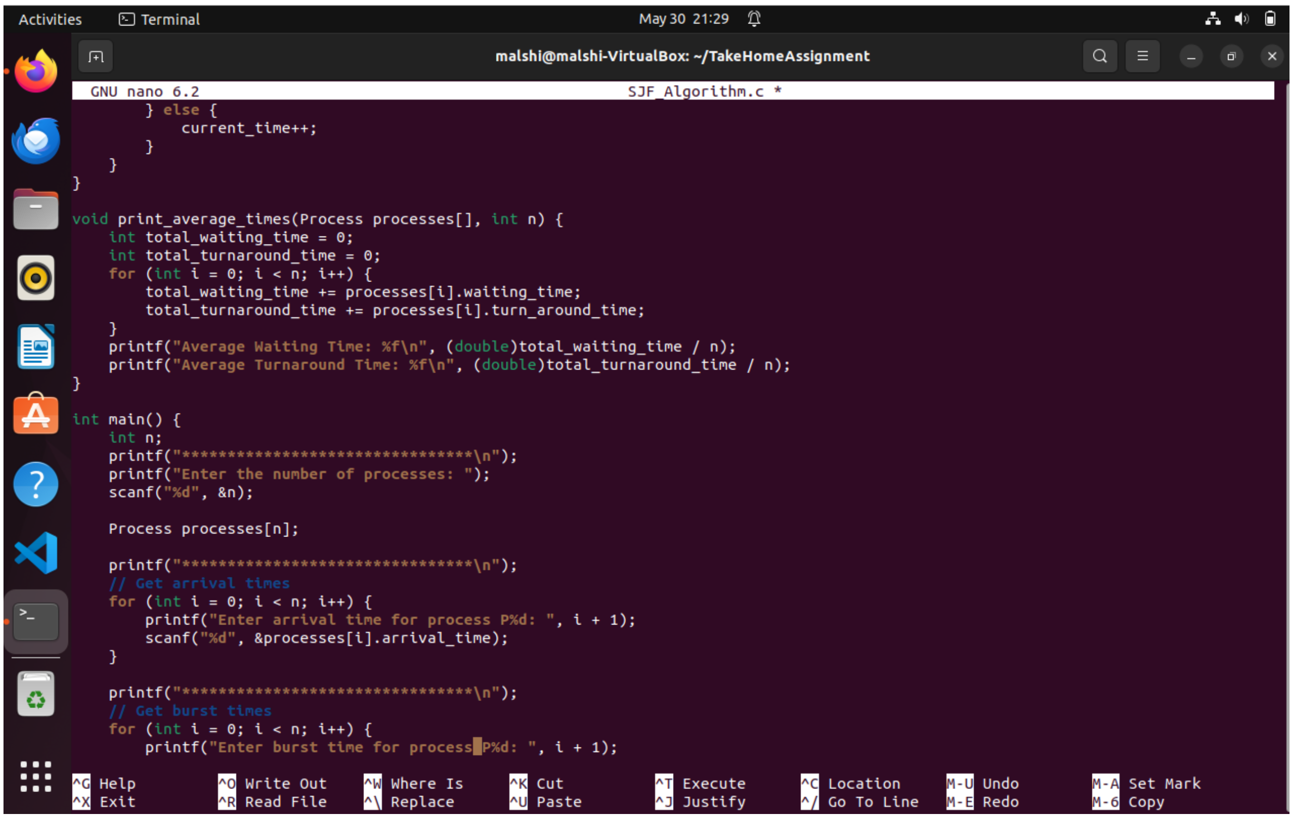


```
  GNU nano 6.2                                       FCFS_Algorithm.c
    }
    printf("Average Waiting Time: %f\n", (double)total_waiting_time / n);
    printf("Average Turnaround Time: %f\n", (double)total_turnaround_time / n);
}

int main() {
    int n;
    printf("*********************************\n");
    printf("Enter the number of processes: ");
    scanf("%d", &n);

    Process processes[n];

    printf("*********************************\n");
    // Get arrival times
    for (int i = 0; i < n; i++) {
        printf("Enter arrival time for process P%d: ", i + 1);
        scanf("%d", &processes[i].arrival_time);
    }

    printf("*********************************\n");
    // Get burst times
    for (int i = 0; i < n; i++) {
        printf("Enter burst time for process P%d: ", i + 1);
        scanf("%d", &processes[i].burst_time);
        processes[i].waiting_time = 0;
        processes[i].turn_around_time = 0;
    }

    printf("*********************************\n");
    calculate_FCFS(processes, n);
    print_average_times(processes, n);

    return 0;
}

^G Help         ^O Write Out    ^W Where Is     ^K Cut          ^T Execute      ^C Location     M-U Undo        M-A Set Mark
^X Exit         ^R Read File    ^\ Replace      ^U Paste        ^J Justify      ^/ Go To Line   M-E Redo        M-6 Copy
```

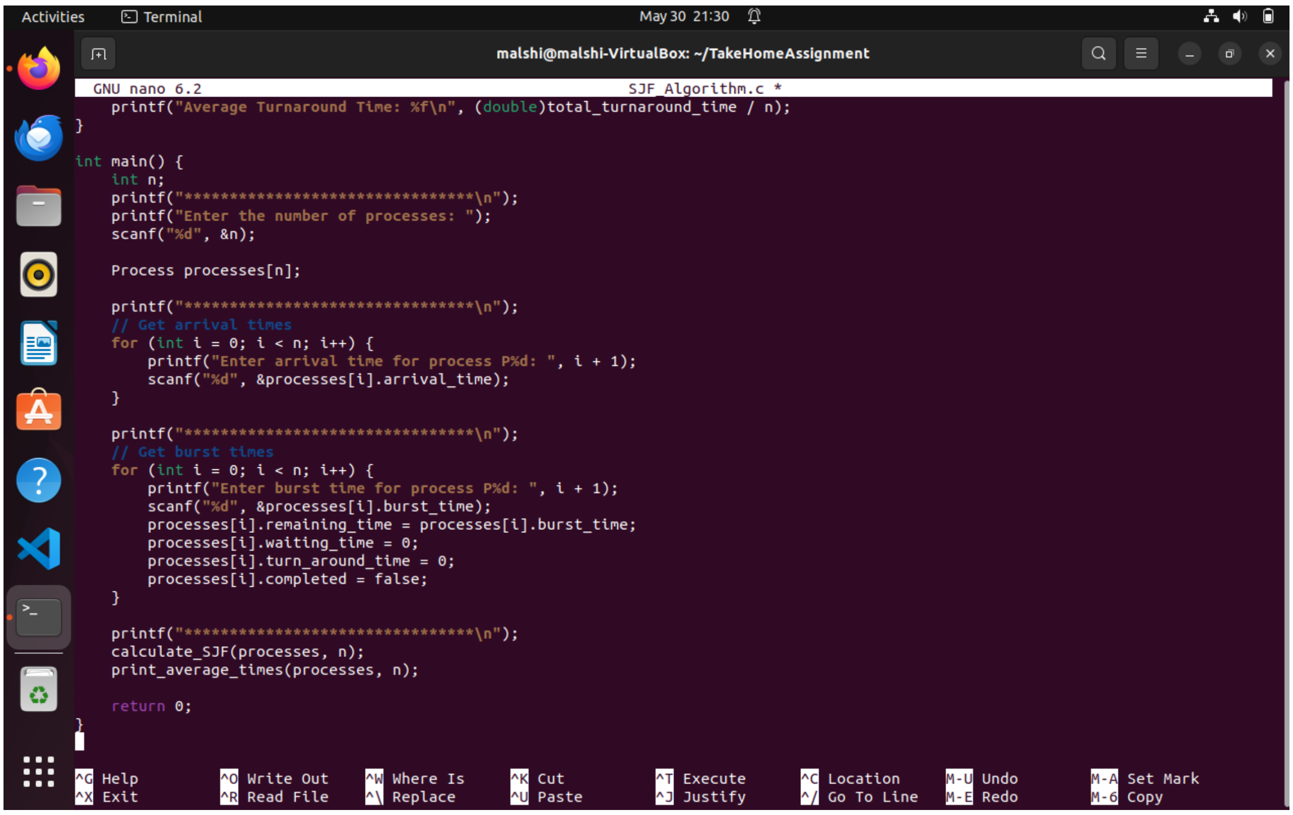
- **SJF Scheduling Algorithm**



```c
#include <stdio.h>
#include <stdbool.h>

typedef struct {
    int arrival_time;
    int burst_time;
    int remaining_time;
    int waiting_time;
    int turn_around_time;
    bool completed;
} Process;

void calculate_SJF(Process processes[], int n) {
    int current_time = 0;
    int completed_processes = 0;
    int min_index;
    bool found;

    while (completed_processes < n) {
        min_index = -1;
        found = false;
        for (int i = 0; i < n; i++) {
            if (processes[i].arrival_time <= current_time && !processes[i].completed) {
                if (!found || processes[i].burst_time < processes[min_index].burst_time) {
                    min_index = i;
                    found = true;
                }
            }
        }

        if (found) {
            current_time += processes[min_index].burst_time;
            processes[min_index].waiting_time = current_time - processes[min_index].burst_time - processes[min_index].arrival_time;
            processes[min_index].turn_around_time = processes[min_index].waiting_time + processes[min_index].burst_time;
            processes[min_index].completed = true;
            completed_processes++;
```

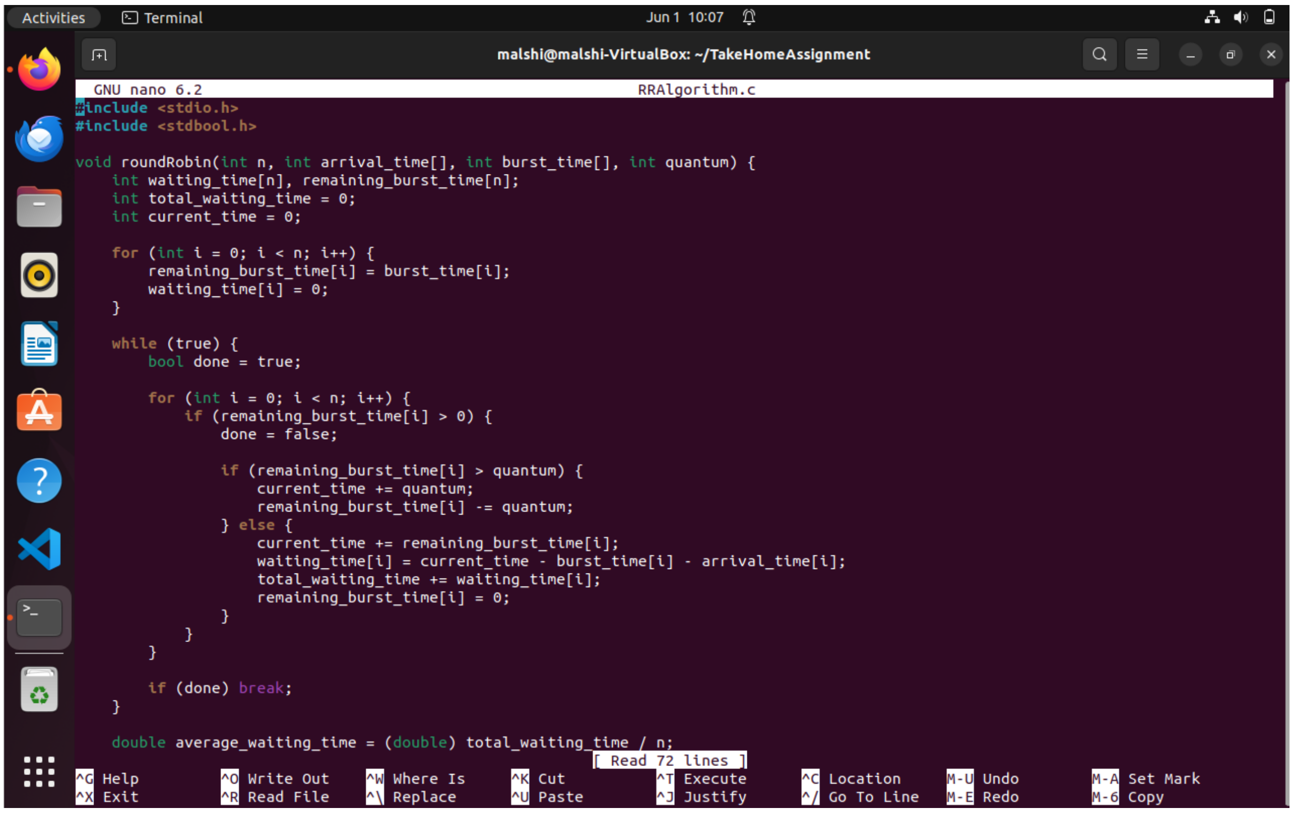


```c
        } else {
            current_time++;
        }
    }
}

void print_average_times(Process processes[], int n) {
    int total_waiting_time = 0;
    int total_turnaround_time = 0;
    for (int i = 0; i < n; i++) {
        total_waiting_time += processes[i].waiting_time;
        total_turnaround_time += processes[i].turn_around_time;
    }
    printf("Average Waiting Time: %f\n", (double)total_waiting_time / n);
    printf("Average Turnaround Time: %f\n", (double)total_turnaround_time / n);
}

int main() {
    int n;
    printf("*********************************\n");
    printf("Enter the number of processes: ");
    scanf("%d", &n);

    Process processes[n];

    printf("*********************************\n");
    // Get arrival times
    for (int i = 0; i < n; i++) {
        printf("Enter arrival time for process P%d: ", i + 1);
        scanf("%d", &processes[i].arrival_time);
    }

    printf("*********************************\n");
    // Get burst times
    for (int i = 0; i < n; i++) {
        printf("Enter burst time for process P%d: ", i + 1);
```

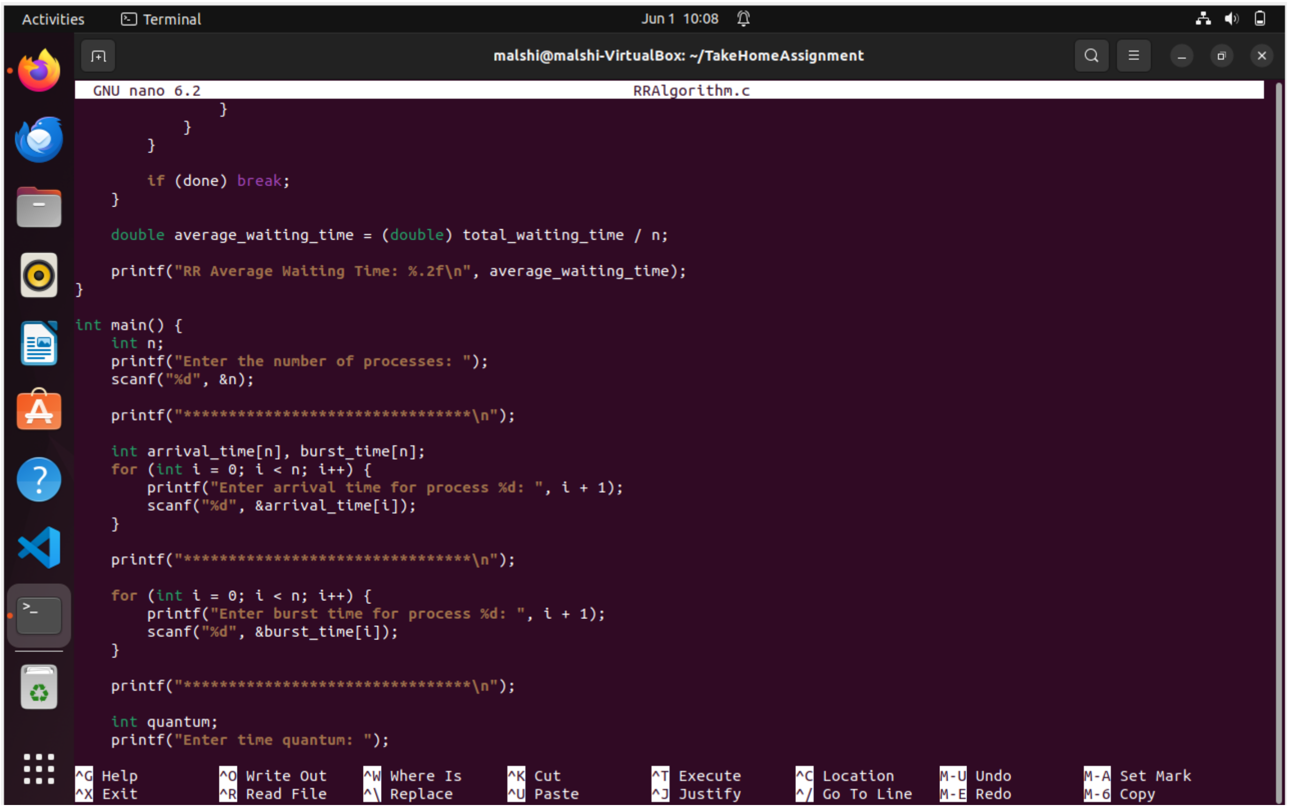


```
Activities    Terminal                    May 30  21:30

malshi@malshi-VirtualBox: ~/TakeHomeAssignment

  GNU nano 6.2                          SJF_Algorithm.c *
    printf("Average Turnaround Time: %f\n", (double)total_turnaround_time / n);
}

int main() {
    int n;
    printf("********************************\n");
    printf("Enter the number of processes: ");
    scanf("%d", &n);

    Process processes[n];

    printf("********************************\n");
    // Get arrival times
    for (int i = 0; i < n; i++) {
        printf("Enter arrival time for process P%d: ", i + 1);
        scanf("%d", &processes[i].arrival_time);
    }

    printf("********************************\n");
    // Get burst times
    for (int i = 0; i < n; i++) {
        printf("Enter burst time for process P%d: ", i + 1);
        scanf("%d", &processes[i].burst_time);
        processes[i].remaining_time = processes[i].burst_time;
        processes[i].waiting_time = 0;
        processes[i].turn_around_time = 0;
        processes[i].completed = false;
    }

    printf("********************************\n");
    calculate_SJF(processes, n);
    print_average_times(processes, n);

    return 0;
}

^G Help       ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo     M-A Set Mark
^X Exit       ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^/ Go To Line M-E Redo     M-6 Copy
```

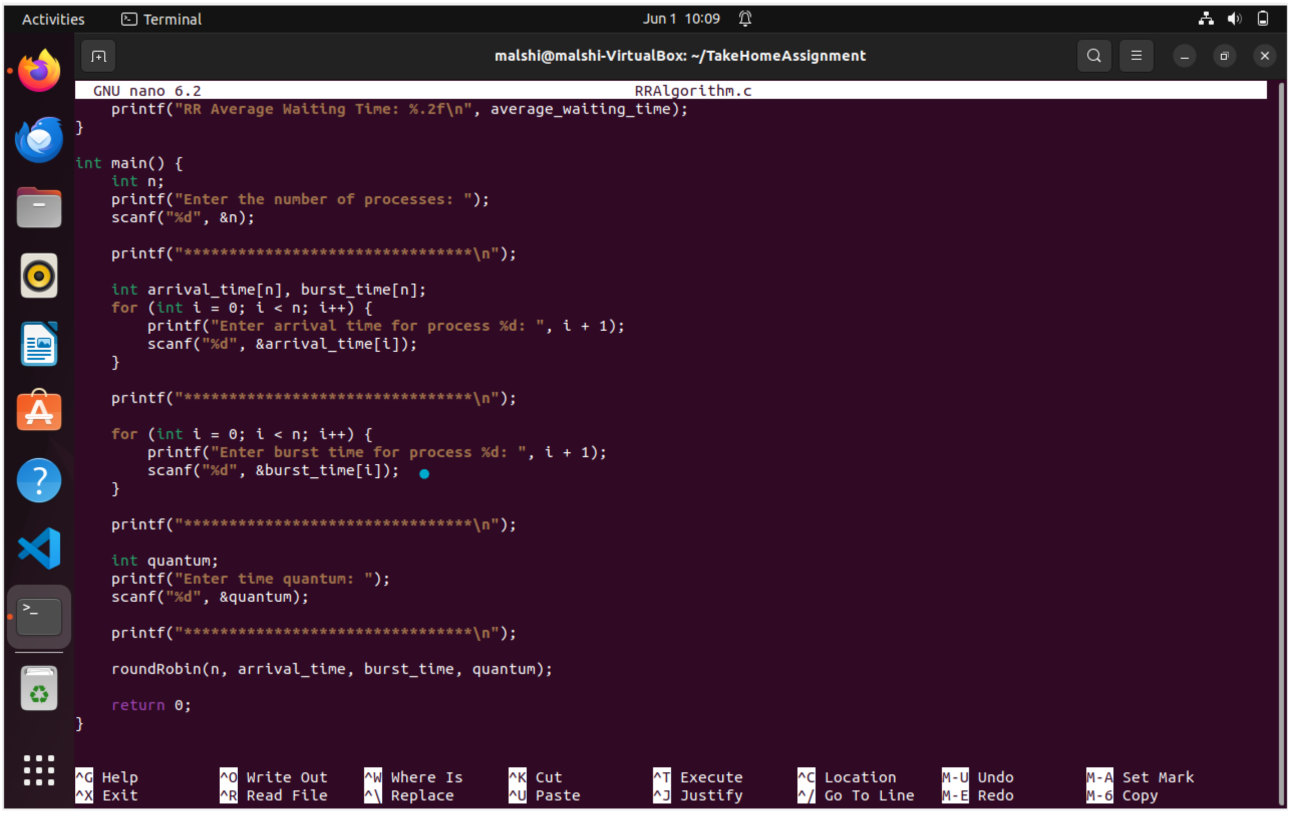
- **RR Scheduling Algorithm**



```
Activities    Terminal                    Jun 1  10:07

malshi@malshi-VirtualBox: ~/TakeHomeAssignment

  GNU nano 6.2                          RRAlgorithm.c
#include <stdio.h>
#include <stdbool.h>

void roundRobin(int n, int arrival_time[], int burst_time[], int quantum) {
    int waiting_time[n], remaining_burst_time[n];
    int total_waiting_time = 0;
    int current_time = 0;

    for (int i = 0; i < n; i++) {
        remaining_burst_time[i] = burst_time[i];
        waiting_time[i] = 0;
    }

    while (true) {
        bool done = true;

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

                if (remaining_burst_time[i] > quantum) {
                    current_time += quantum;
                    remaining_burst_time[i] -= quantum;
                } else {
                    current_time += remaining_burst_time[i];
                    waiting_time[i] = current_time - burst_time[i] - arrival_time[i];
                    total_waiting_time += waiting_time[i];
                    remaining_burst_time[i] = 0;
                }
            }
        }

        if (done) break;
    }

    double average_waiting_time = (double) total_waiting_time / n;
                                                  [ Read 72 lines ]
^G Help       ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo     M-A Set Mark
^X Exit       ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^/ Go To Line M-E Redo     M-6 Copy
```



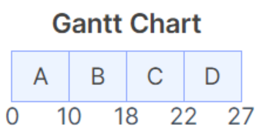
```
  GNU nano 6.2                                        RRAlgorithm.c
                }
            }
        }

        if (done) break;
    }

    double average_waiting_time = (double) total_waiting_time / n;

    printf("RR Average Waiting Time: %.2f\n", average_waiting_time);
}

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

    printf("**********************************\n");

    int arrival_time[n], burst_time[n];
    for (int i = 0; i < n; i++) {
        printf("Enter arrival time for process %d: ", i + 1);
        scanf("%d", &arrival_time[i]);
    }

    printf("**********************************\n");

    for (int i = 0; i < n; i++) {
        printf("Enter burst time for process %d: ", i + 1);
        scanf("%d", &burst_time[i]);
    }

    printf("**********************************\n");

    int quantum;
    printf("Enter time quantum: ");
^G Help        ^O Write Out   ^W Where Is    ^K Cut         ^T Execute     ^C Location    M-U Undo       M-A Set Mark
^X Exit        ^R Read File   ^\ Replace     ^U Paste       ^J Justify     ^/ Go To Line  M-E Redo       M-6 Copy
```



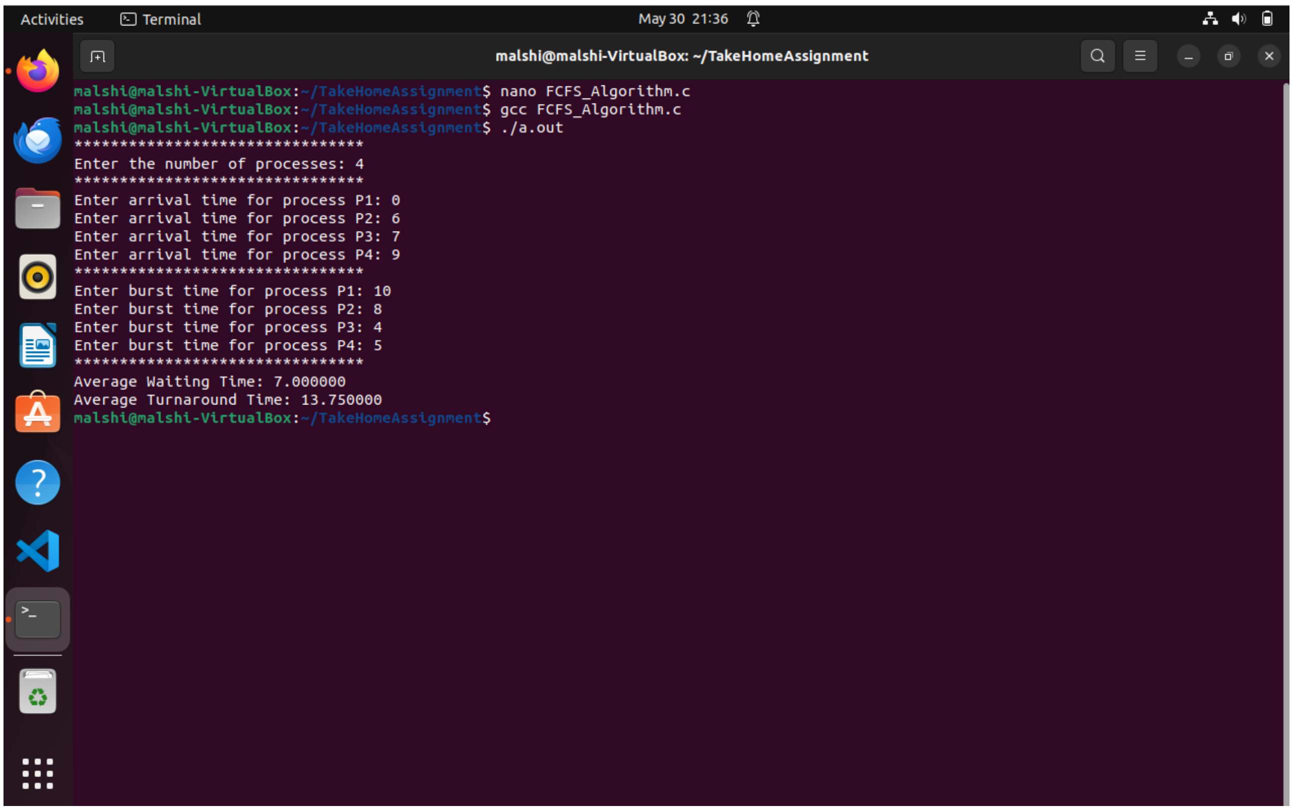
```
  GNU nano 6.2                                        RRAlgorithm.c
    printf("RR Average Waiting Time: %.2f\n", average_waiting_time);
}

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

    printf("**********************************\n");

    int arrival_time[n], burst_time[n];
    for (int i = 0; i < n; i++) {
        printf("Enter arrival time for process %d: ", i + 1);
        scanf("%d", &arrival_time[i]);
    }

    printf("**********************************\n");

    for (int i = 0; i < n; i++) {
        printf("Enter burst time for process %d: ", i + 1);
        scanf("%d", &burst_time[i]);
    }

    printf("**********************************\n");

    int quantum;
    printf("Enter time quantum: ");
    scanf("%d", &quantum);

    printf("**********************************\n");

    roundRobin(n, arrival_time, burst_time, quantum);

    return 0;
}
^G Help        ^O Write Out   ^W Where Is    ^K Cut         ^T Execute     ^C Location    M-U Undo       M-A Set Mark
^X Exit        ^R Read File   ^\ Replace     ^U Paste       ^J Justify     ^/ Go To Line  M-E Redo       M-6 Copy
```

Output demonstrating the execution of processes using each algorithm

- **FCFS Scheduling Algorithm**

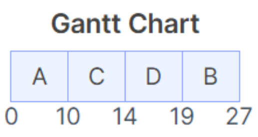


Job	Arrival Time	Burst Time	Finish Time	Turnaround Time	Waiting Time
A	0	10	10	10	0
B	6	8	18	12	4
C	7	4	22	15	11
D	9	5	27	18	13
			Average	55 / 4 = 13.75	28 / 4 = 7

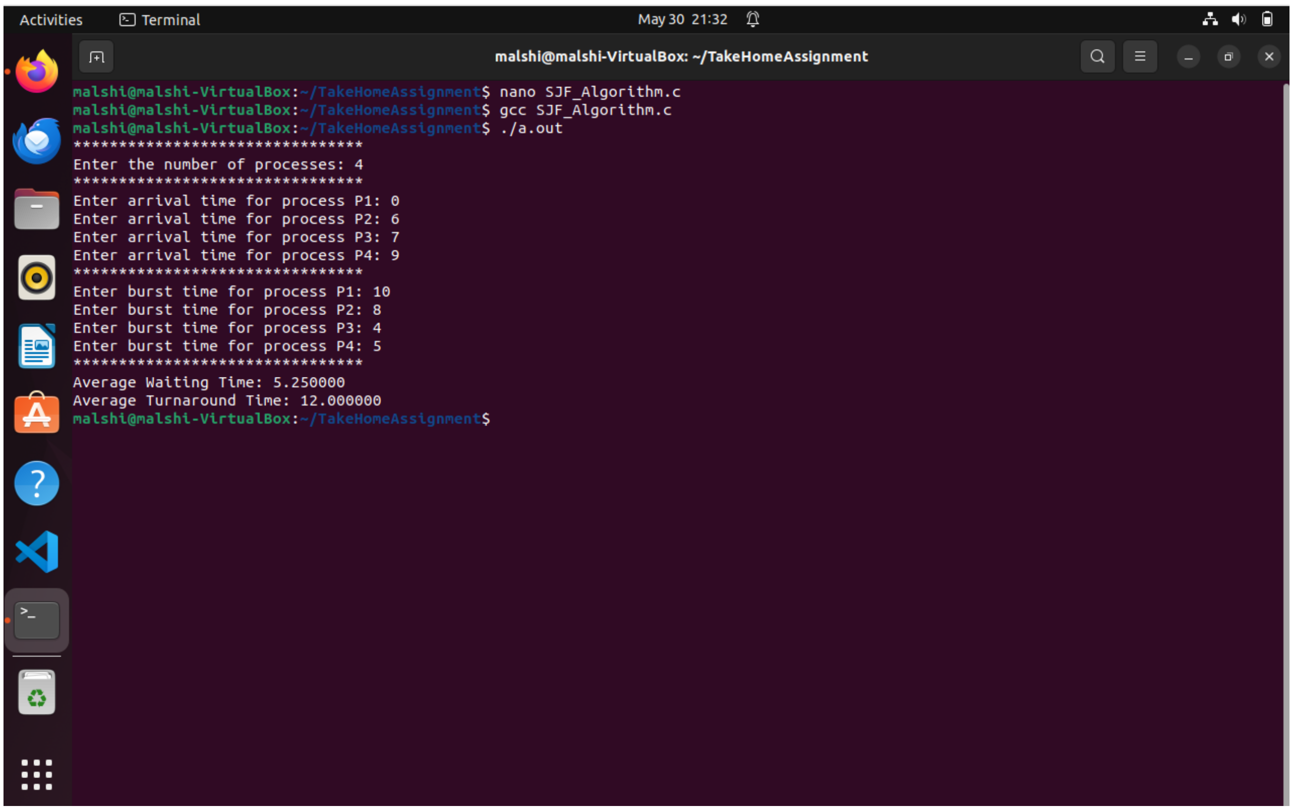


```
malshi@malshi-VirtualBox:~/TakeHomeAssignment$ nano FCFS_Algorithm.c
malshi@malshi-VirtualBox:~/TakeHomeAssignment$ gcc FCFS_Algorithm.c
malshi@malshi-VirtualBox:~/TakeHomeAssignment$ ./a.out
********************************
Enter the number of processes: 4
********************************
Enter arrival time for process P1: 0
Enter arrival time for process P2: 6
Enter arrival time for process P3: 7
Enter arrival time for process P4: 9
********************************
Enter burst time for process P1: 10
Enter burst time for process P2: 8
Enter burst time for process P3: 4
Enter burst time for process P4: 5
********************************
Average Waiting Time: 7.000000
Average Turnaround Time: 13.750000
malshi@malshi-VirtualBox:~/TakeHomeAssignment$
```

- **SJF Scheduling Algorithm**



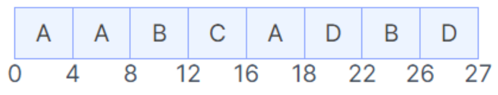
Job	Arrival Time	Burst Time	Finish Time	Turnaround Time	Waiting Time
A	0	10	10	10	0
B	6	8	27	21	13
C	7	4	14	7	3
D	9	5	19	10	5
			Average	48 / 4 = 12	21 / 4 = 5.25

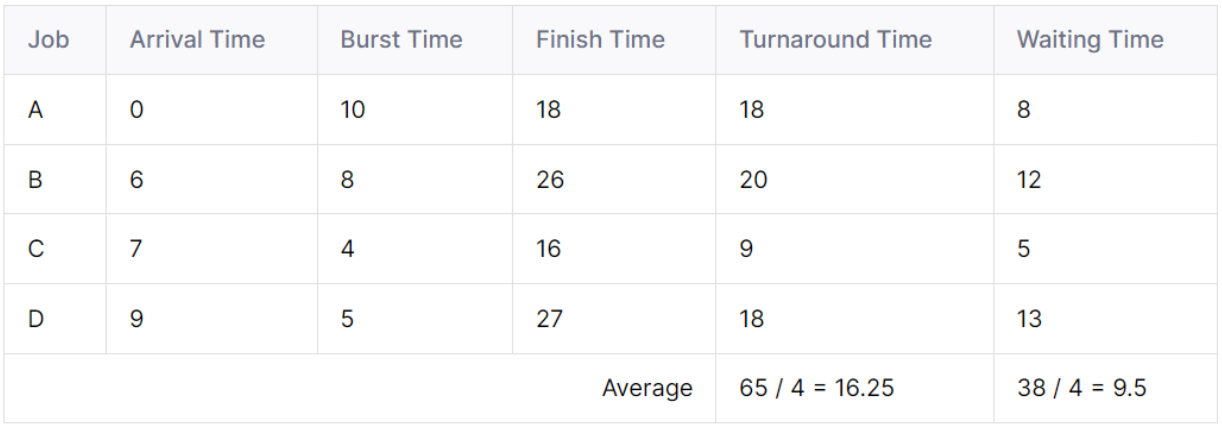


```
malshi@malshi-VirtualBox:~/TakeHomeAssignment$ nano SJF_Algorithm.c
malshi@malshi-VirtualBox:~/TakeHomeAssignment$ gcc SJF_Algorithm.c
malshi@malshi-VirtualBox:~/TakeHomeAssignment$ ./a.out
********************************
Enter the number of processes: 4
********************************
Enter arrival time for process P1: 0
Enter arrival time for process P2: 6
Enter arrival time for process P3: 7
Enter arrival time for process P4: 9
********************************
Enter burst time for process P1: 10
Enter burst time for process P2: 8
Enter burst time for process P3: 4
Enter burst time for process P4: 5
********************************
Average Waiting Time: 5.250000
Average Turnaround Time: 12.000000
malshi@malshi-VirtualBox:~/TakeHomeAssignment$
```

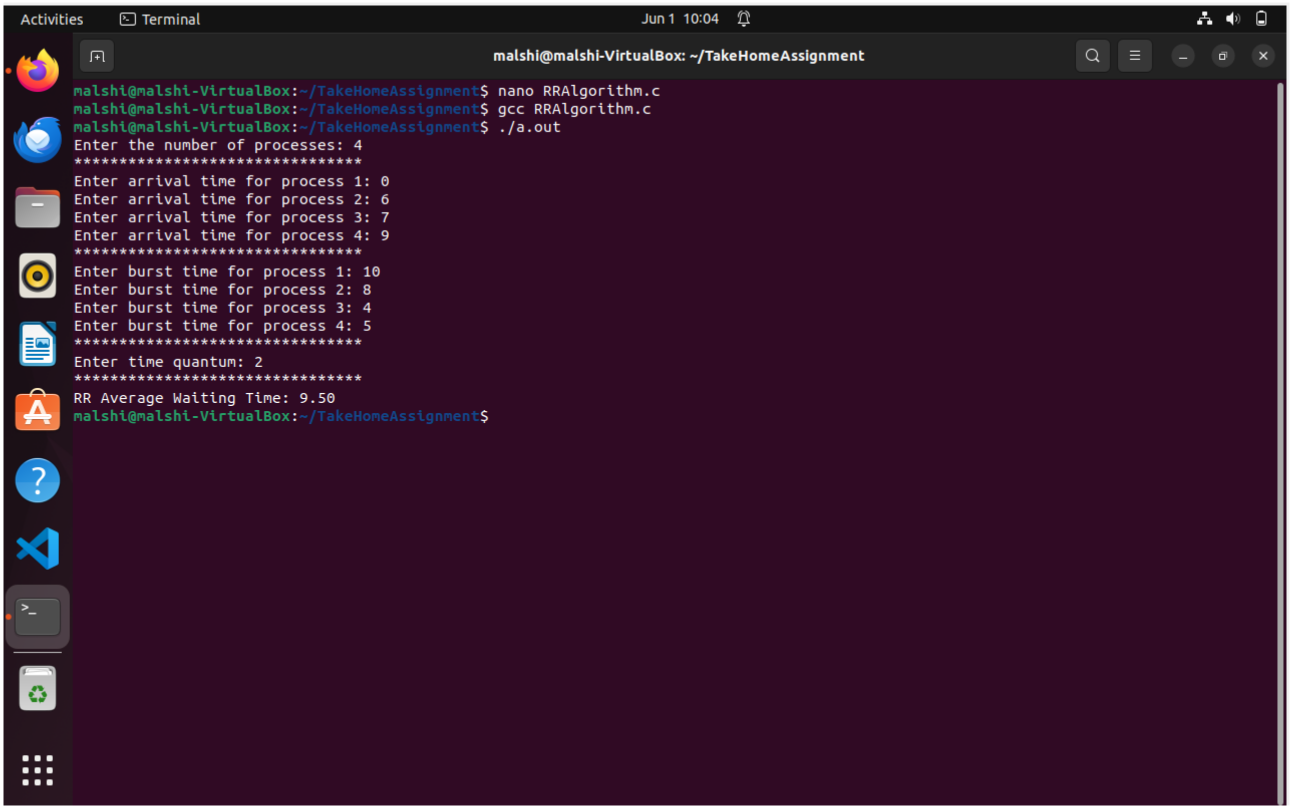
- **RR Scheduling Algorithm**

Gantt Chart





Job	Arrival Time	Burst Time	Finish Time	Turnaround Time	Waiting Time
A	0	10	18	18	8
B	6	8	26	20	12
C	7	4	16	9	5
D	9	5	27	18	13
			Average	65 / 4 = 16.25	38 / 4 = 9.5



```
malshi@malshi-VirtualBox:~/TakeHomeAssignment$ nano RRAlgorithm.c
malshi@malshi-VirtualBox:~/TakeHomeAssignment$ gcc RRAlgorithm.c
malshi@malshi-VirtualBox:~/TakeHomeAssignment$ ./a.out
Enter the number of processes: 4
********************************
Enter arrival time for process 1: 0
Enter arrival time for process 2: 6
Enter arrival time for process 3: 7
Enter arrival time for process 4: 9
********************************
Enter burst time for process 1: 10
Enter burst time for process 2: 8
Enter burst time for process 3: 4
Enter burst time for process 4: 5
********************************
Enter time quantum: 2
********************************
RR Average Waiting Time: 9.50
malshi@malshi-VirtualBox:~/TakeHomeAssignment$
```

Analysis comparing the performance of the algorithms based on average waiting time

First Come First Serve (FCFS) Average Waiting Time: 7 units
Shortest Job First (SJF) Average Waiting Time: 5.25 units
Round Robin (RR) Average Waiting Time: 9.5 units

First Come First Serve (FCFS) has an average waiting time of 7 units, which is higher than SJF but lower than RR. FCFS can cause the convoy effect, where shorter jobs wait for longer ones, increasing the average waiting time.

Shortest Job First (SJF) has the lowest average waiting time of 5.25 units. This is expected as SJF minimizes the waiting time by prioritizing shorter jobs, leading to better performance in terms of waiting time.

Round Robin (RR) has the highest average waiting time of 9.5 units. RR is designed to be fair and provide time slices to each job, but this can lead to higher waiting times, especially if the time quantum is not optimized. The frequent context switching in RR can also contribute to increased waiting times.

SJF performs best in terms of minimizing average waiting time.
FCFS offers moderate performance but can be inefficient for longer jobs.
RR provides fairness but at the cost of higher average waiting time.

Each algorithm has its trade-offs, and the choice depends on the specific requirements of the scheduling environment, such as the need for fairness (RR) versus minimizing waiting time (SJF).