Reg No :21BCB0107

Name: Shivam Dave

Slot: B2+TB2

Exercise 2 – CPU Scheduling Algorithm

Write a C program to simulate the following non-pre-emptive CPU scheduling algorithms to find turnaround time and waiting time.

a) FCFS

Aim: to implement CPU scheduling using FCFS

Algorithm:

1. Start by initializing the arrival time and burst time for each process.
2. Sort the processes based on their arrival time. If two or more processes have the same arrival time, maintain their original order.
3. Set the current time (initially 0) and select the process with the smallest arrival time.
4. Execute the selected process until its burst time is completed.
5. Update the current time to the completion time of the executed process.
6. Repeat steps 4 and 5 for all the remaining processes in the order they arrived.
7. Calculate the waiting time for each process by subtracting its arrival time from its completion time.
8. Calculate the turnaround time for each process by subtracting its arrival time from its completion time.
9. Calculate the average waiting time by summing up the waiting times of all processes and dividing by the total number of processes.
10. Calculate the average turnaround time by summing up the turnaround times of all processes and dividing by the total number of processes.
11. Print the waiting time and turnaround time for each process, as well as the average waiting time and average turnaround time.

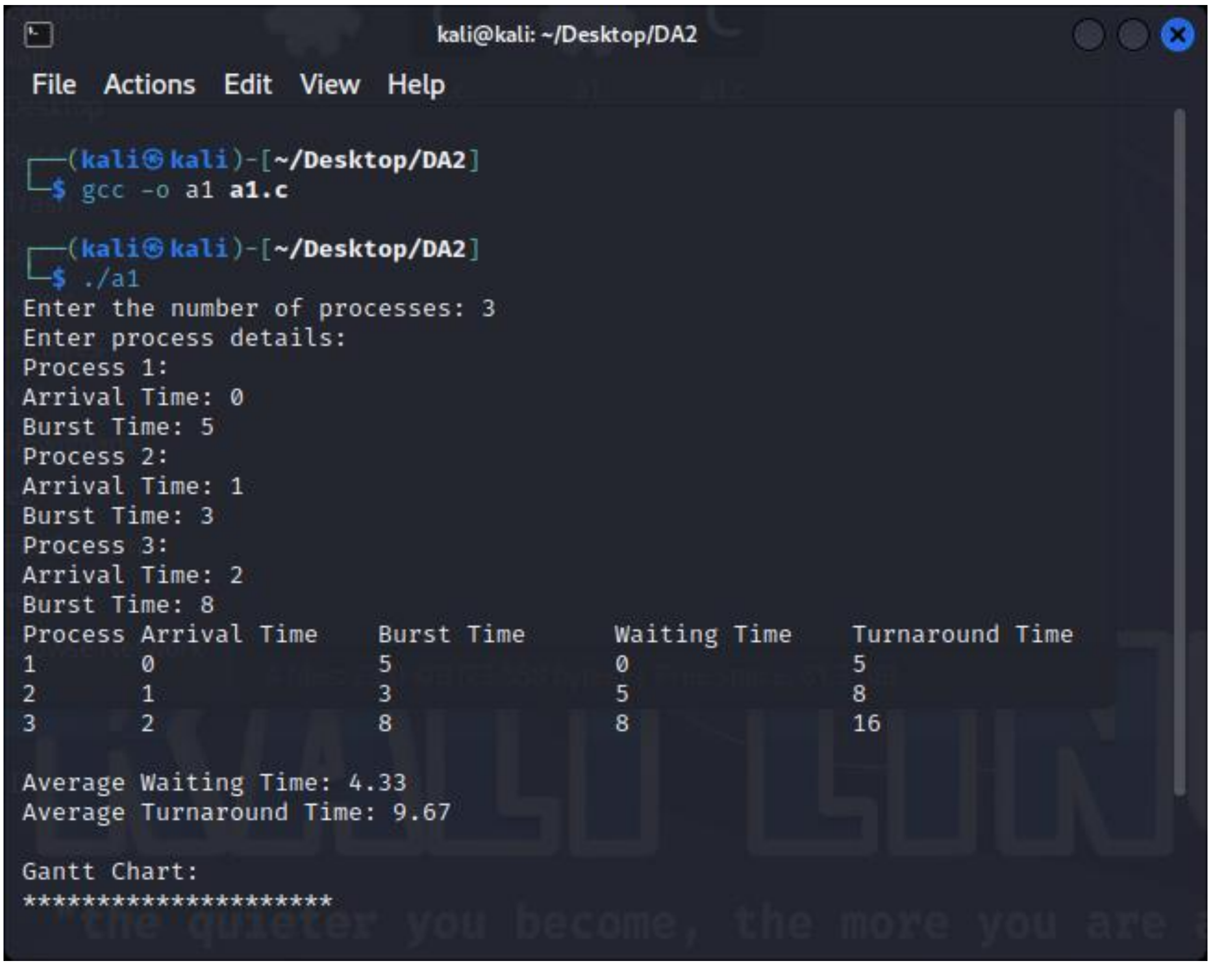
Source Code:

```c
#include <stdio.h>

//structure to represent the process
struct Process
{
    int processID;       // Process ID
    int arrivalTime;     // Arrival time of the process
    int burstTime;       // Burst time of the process
};

// Creating a function to calculate waiting time & turnaround time for each
process
```

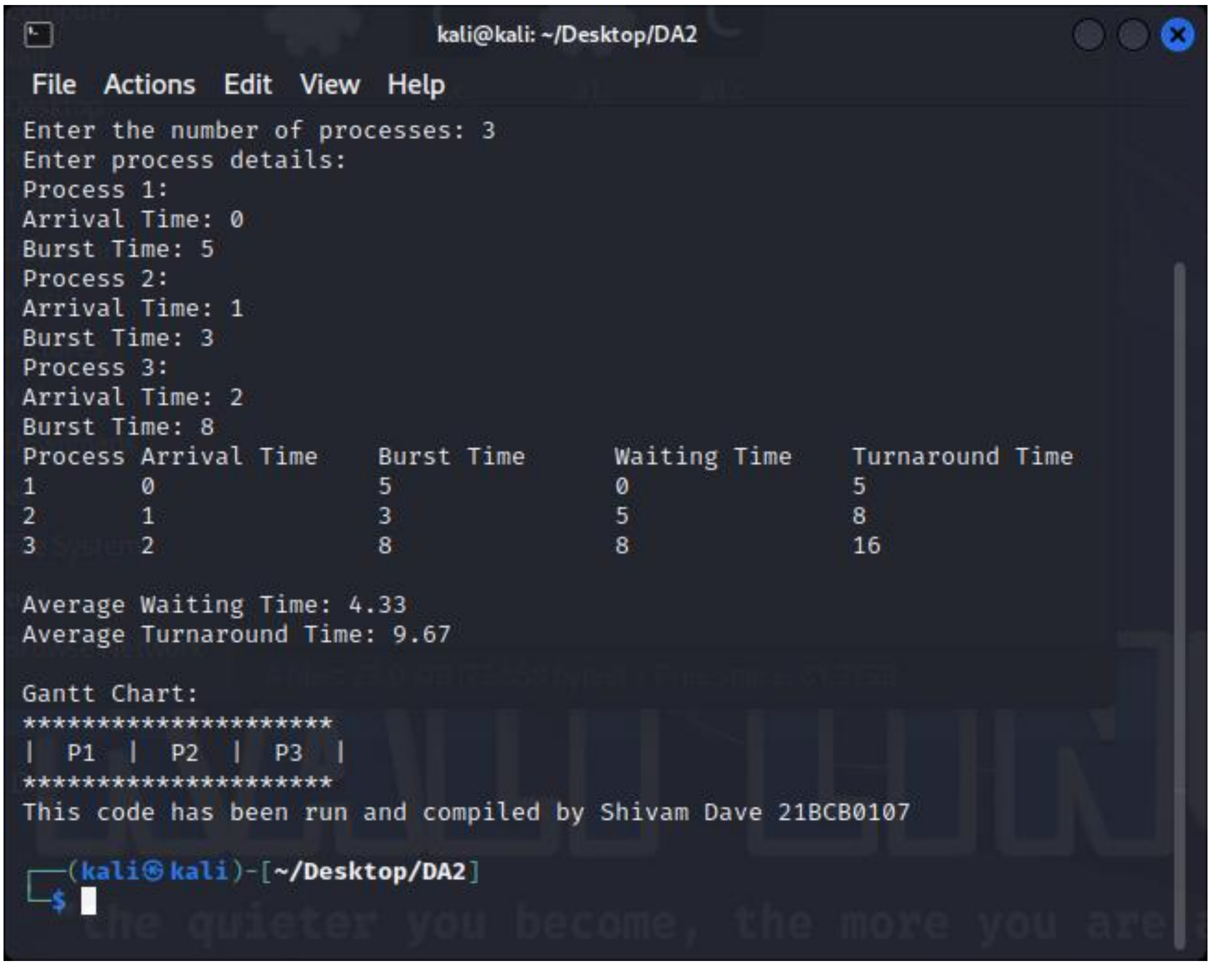
```
void calculateTimes(struct Process processes[], int n, int waitingTime[], int
turnaroundTime[])
{
    int i;
    // We know that the waiting time for the first process is 0
    waitingTime[0] = 0;

    // To calculate waiting time for each process
    for (i = 1; i < n; i++)
    {
        // We know that the Waiting-time = Burst-time of previous process +
Waiting-time of previous process
        waitingTime[i] = waitingTime[i - 1] + processes[i - 1].burstTime;
    }

    // To calculate turnaround time for each process
    for (i = 0; i < n; i++)
    {
        // As Turnaround time = Waiting time + Burst time
        turnaroundTime[i] = waitingTime[i] + processes[i].burstTime;
    }
}

// Creating a function to calculate average waiting time and average
turnaround time
void calculateAverageTimes(struct Process processes[], int n) // Where n is
the number of processes
{
    //Initializing
    int i, totalWaitingTime = 0, totalTurnaroundTime = 0;
    int waitingTime[n], turnaroundTime[n];

    // We have to now calculate waiting time and turnaround time for each
process
    calculateTimes(processes, n, waitingTime, turnaroundTime);

    // To calculate total waiting time and total turnaround time
    for (i = 0; i < n; i++)
    {
        totalWaitingTime += waitingTime[i];
        /* "+=" this means total waiting time = total waiting time + waiting
time for the ith process*/
        totalTurnaroundTime += turnaroundTime[i];
    }

    // Now to calculate average waiting time and average turnaround time we
divide by n(no. of processes)
    float avgWaitingTime = (float)totalWaitingTime / n;
```

```c
    float avgTurnaroundTime = (float)totalTurnaroundTime / n;

    // Printing the results
    printf("Process\tArrival Time\tBurst Time\tWaiting Time\tTurnaround
Time\n");
    for (i = 0; i < n; i++)
    {
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\n", processes[i].processID,
processes[i].arrivalTime,
               processes[i].burstTime, waitingTime[i], turnaroundTime[i]);
    }
    printf("\nAverage Waiting Time: %.2f\n", avgWaitingTime);
    printf("Average Turnaround Time: %.2f\n", avgTurnaroundTime);

        // Print the Gantt chart
    printf("\nGantt Chart:\n");
    for (i = 0; i < n; i++) {
        printf("******");
    }
    printf("\n|");

    for (i = 0; i < n; i++) {
        printf("  P%d  ", processes[i].processID);
        printf("|");
    }
    printf("\n");

    for (i = 0; i < n; i++) {
        printf("*******");
    }
    printf("\n");
}

int main()
{
    int n, i;
    printf("Enter the number of processes: ");// taking in no. of processes
    scanf("%d", &n);

    // Create an array of processes
    struct Process processes[n];

    // Inputing the process details from the user step by step
    printf("Enter process details:\n");
    for (i = 0; i < n; i++)
    {
        printf("Process %d:\n", i + 1);
        printf("Arrival Time: ");
        //Inputing the arrival time
        scanf("%d", &processes[i].arrivalTime);
        printf("Burst Time: ");
        //Inputing the burst time
        scanf("%d", &processes[i].burstTime);
        processes[i].processID = i + 1;//incrementing proccessID
    }

    // Calculating average waiting time and average turnaround time
    calculateAverageTimes(processes, n);
    printf("This code has been run and compiled by Shivam Dave 21BCB0107");
    return 0;
}
```

Output and Result:



```
┌──(kali㉿kali)-[~/Desktop/DA2]
└─$ gcc -o a1 a1.c

┌──(kali㉿kali)-[~/Desktop/DA2]
└─$ ./a1
Enter the number of processes: 3
Enter process details:
Process 1:
Arrival Time: 0
Burst Time: 5
Process 2:
Arrival Time: 1
Burst Time: 3
Process 3:
Arrival Time: 2
Burst Time: 8
Process Arrival Time    Burst Time      Waiting Time    Turnaround Time
1       0               5               0               5
2       1               3               5               8
3       2               8               8               16

Average Waiting Time: 4.33
Average Turnaround Time: 9.67

Gantt Chart:
*********************
```



```
Enter the number of processes: 3
Enter process details:
Process 1:
Arrival Time: 0
Burst Time: 5
Process 2:
Arrival Time: 1
Burst Time: 3
Process 3:
Arrival Time: 2
Burst Time: 8
Process Arrival Time     Burst Time      Waiting Time     Turnaround Time
1       0                5               0                5
2       1                3               5                8
3       2                8               8                16

Average Waiting Time: 4.33
Average Turnaround Time: 9.67

Gantt Chart:
*********************
|  P1  |  P2  |  P3  |
*********************
This code has been run and compiled by Shivam Dave 21BCB0107

┌──(kali㉿kali)-[~/Desktop/DA2]
└─$
```

b) SJF

Aim: To implement CPU Shortest Job scheduling

Algorithm:

1. Initialize the ready queue. The ready queue is a list of all processes that are ready to be executed.
2. Find the process with the shortest burst time. The process with the shortest burst time is the process that will take the least amount of time to execute.
3. Remove the process from the ready queue and start executing it. The CPU will start executing the process with the shortest burst time.
4. Repeat steps 2 and 3 until all processes have been executed.

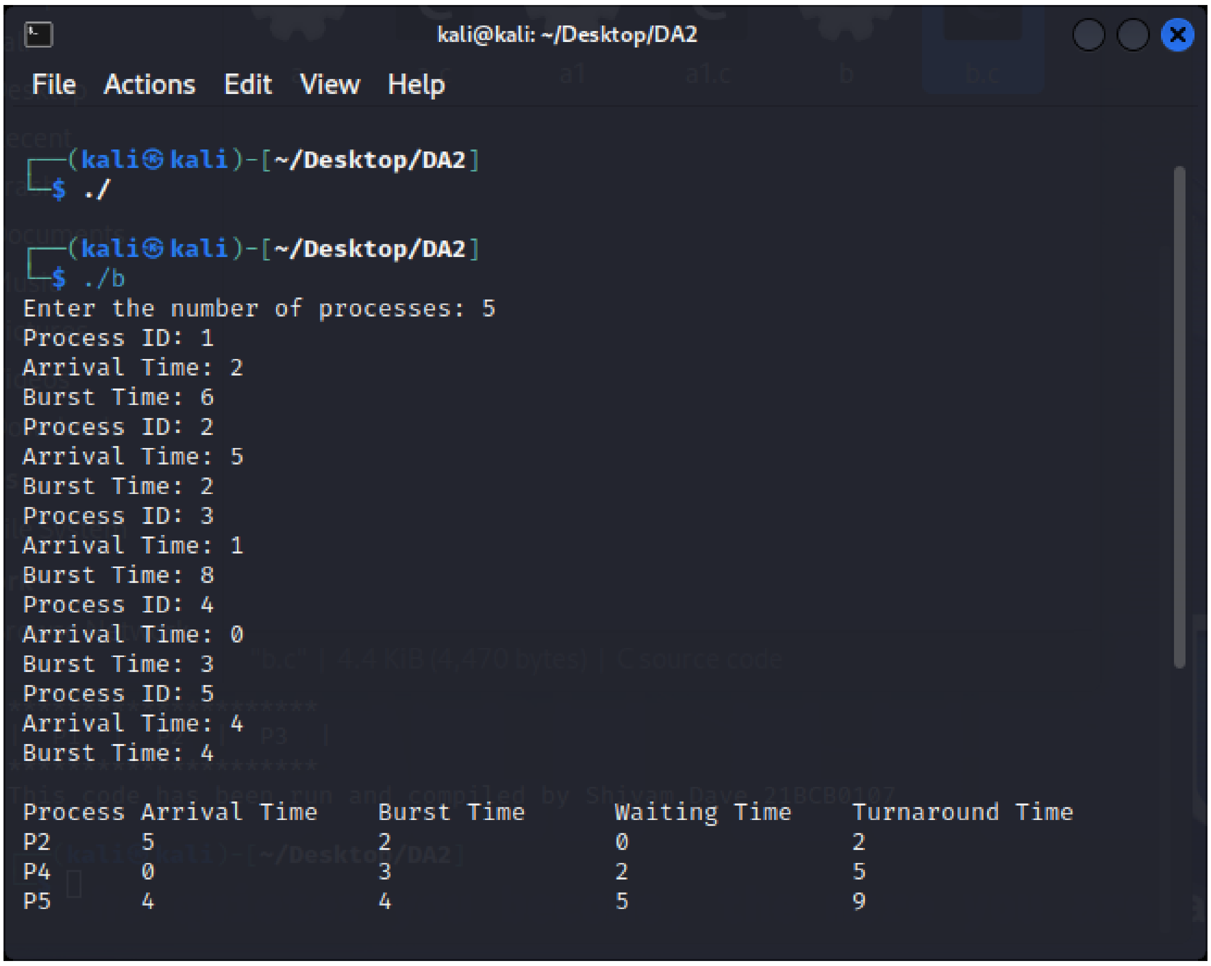
Source Code:

```c
#include <stdio.h>
// SJF CPU algorithm
struct Process
{
    int processID;       // Process ID
    int arrivalTime;     // Arrival time of the process
    int burstTime;       // Burst time of the process
};
```

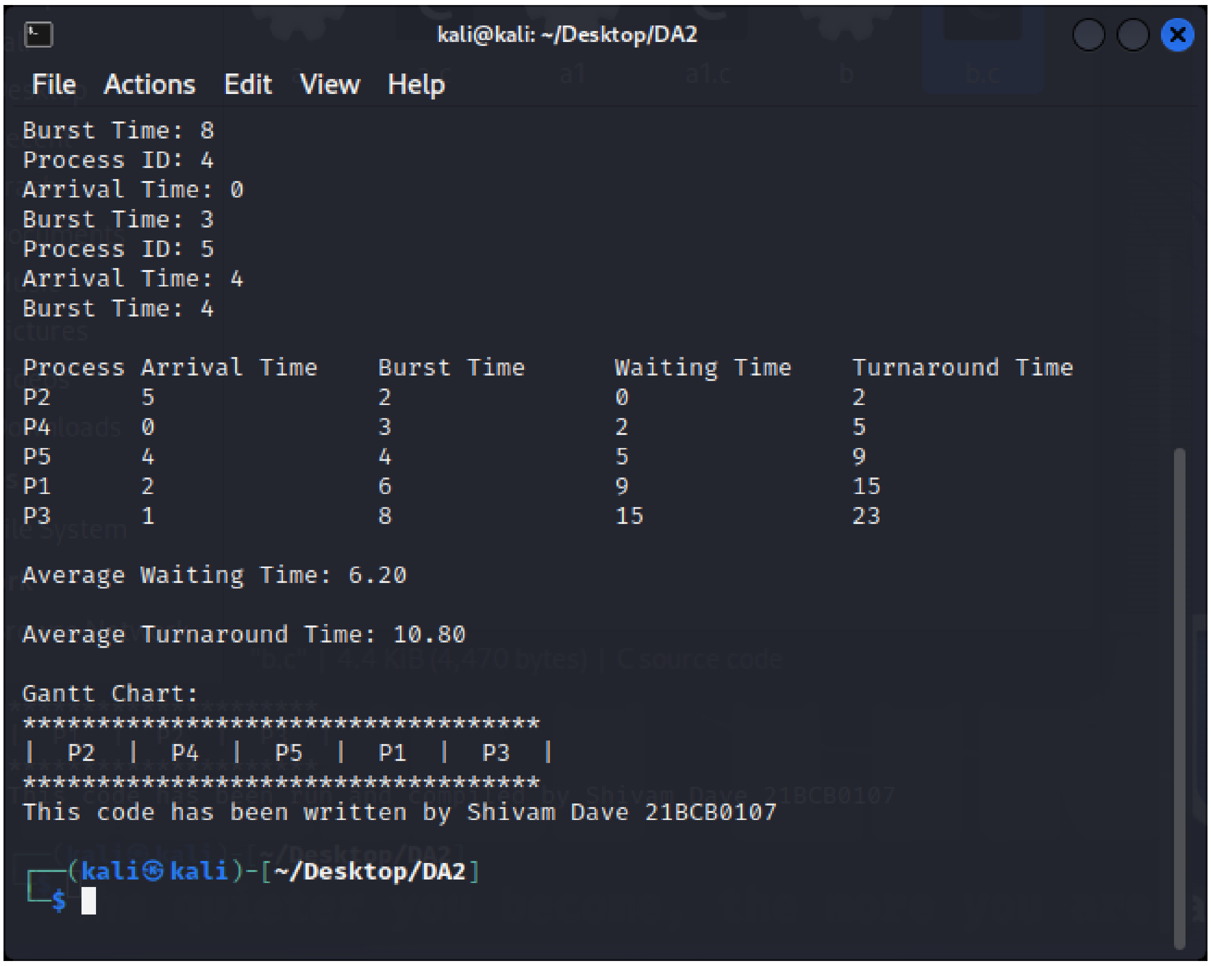
```c
// Creating a function to calculate waiting time & turnaround time for each
process
void calculateTimes(struct Process processes[], int n, int waitingTime[], int
turnaroundTime[])
// Here n is the total number of processes
{
    int i, j, minIndex, temp;//initializing

    // Sorting of the processes based on their burst time in ascending order
using selection sort or any other sorting algo
    for (i = 0; i < n - 1; i++)
    {
        minIndex = i;
        for (j = i + 1; j < n; j++)
        {
            if (processes[j].burstTime < processes[minIndex].burstTime)
            {
                minIndex = j;
            }
        }

        // Swapping of the processes
        temp = processes[minIndex].burstTime;
        processes[minIndex].burstTime = processes[i].burstTime;
        processes[i].burstTime = temp;

        temp = processes[minIndex].arrivalTime;
        processes[minIndex].arrivalTime = processes[i].arrivalTime;
        processes[i].arrivalTime = temp;

        temp = processes[minIndex].processID;
        processes[minIndex].processID = processes[i].processID;
        processes[i].processID = temp;
    }

    // We know that the waiting time for the first process is 0
    waitingTime[0] = 0;

    // To calculate waiting time for each process
    for (i = 1; i < n; i++)
    {
        // We know that the Waiting-time = Burst-time of previous process +
Waiting-time of previous process
        waitingTime[i] = waitingTime[i - 1] + processes[i - 1].burstTime;
    }

    // To calculate turnaround time for each process
    for (i = 0; i < n; i++) {
        // As Turnaround time = Waiting time + Burst time
        turnaroundTime[i] = waitingTime[i] + processes[i].burstTime;
    }
}

// Creating a function to calculate average waiting time and average
turnaround time
void calculateAverageTimes(struct Process processes[], int n)
{
    //Initializing
    int i, totalWaitingTime = 0, totalTurnaroundTime = 0;
    int waitingTime[n], turnaroundTime[n];

    // We have to now calculate waiting time and turnaround time for each
process
    calculateTimes(processes, n, waitingTime, turnaroundTime);

     // To calculate total waiting time and total turnaround time
    for (i = 0; i < n; i++)
    {
        totalWaitingTime += waitingTime[i];
        /* "+=" this means total waiting time = total waiting time + waiting
time for the ith process*/
        totalTurnaroundTime += turnaroundTime[i];
    }

    // Now to calculate average waiting time and average turnaround time we
divide by n(no. of processes)
    float avgWaitingTime = (float)totalWaitingTime / n;
    float avgTurnaroundTime = (float)totalTurnaroundTime / n;

    // Printing the results or the process details
    printf("\nProcess\tArrival Time\tBurst Time\tWaiting Time\tTurnaround
Time\n");
    for (i = 0; i < n; i++)
    {
        printf("P%d\t%d\t\t%d\t\t%d\t\t%d\n", processes[i].processID,
processes[i].arrivalTime,
               processes[i].burstTime, waitingTime[i], turnaroundTime[i]);
    }
    printf("\nAverage Waiting Time: %.2f\n", avgWaitingTime);
    printf("\nAverage Turnaround Time: %.2f\n", avgTurnaroundTime);

    // Printing the Gantt chart
    printf("\nGantt Chart:\n");
    for (i = 0; i < n; i++)
    {
        printf("*******");
    }
    printf("\n|");

    for (i = 0; i < n; i++)
    {
        printf("  P%d  ", processes[i].processID);
        printf("|");
    }
    printf("\n");

    for (i = 0; i < n; i++)
    {
        printf("*******");
    }
    printf("\n");
}

int main()
{
    //Initializing
    int n, i;

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

    // Create an array of processes
    struct Process processes[n];

    // Inputing the process details from the user step by step
    for (i = 0; i < n; i++) {
        printf("Process ID: ");
        scanf("%d", &processes[i].processID);
        printf("Arrival Time: ");
        scanf("%d", &processes[i].arrivalTime);
        printf("Burst Time: ");
        scanf("%d", &processes[i].burstTime);
    }

    // Calculating average waiting time and average turnaround time
    calculateAverageTimes(processes, n);
    printf("This code has been written by Shivam Dave 21BCB0107");
    return 0;
}
```

21BCB0107

Output And Result:



```
┌──(kali㉿kali)-[~/Desktop/DA2]
└─$ ./

┌──(kali㉿kali)-[~/Desktop/DA2]
└─$ ./b
Enter the number of processes: 5
Process ID: 1
Arrival Time: 2
Burst Time: 6
Process ID: 2
Arrival Time: 5
Burst Time: 2
Process ID: 3
Arrival Time: 1
Burst Time: 8
Process ID: 4
Arrival Time: 0
Burst Time: 3
Process ID: 5
Arrival Time: 4
Burst Time: 4

Process Arrival Time    Burst Time      Waiting Time    Turnaround Time
P2      5               2               0               2
P4      0               3               2               5
P5      4               4               5               9
```



```
Burst Time: 8
Process ID: 4
Arrival Time: 0
Burst Time: 3
Process ID: 5
Arrival Time: 4
Burst Time: 4

Process Arrival Time    Burst Time      Waiting Time    Turnaround Time
P2      5               2               0               2
P4      0               3               2               5
P5      4               4               5               9
P1      2               6               9               15
P3      1               8               15              23

Average Waiting Time: 6.20

Average Turnaround Time: 10.80

Gantt Chart:
***********************************
|  P2  |  P4  |  P5  |  P1  |  P3  |
***********************************
This code has been written by Shivam Dave 21BCB0107

┌──(kali㉿kali)-[~/Desktop/DA2]
└─$ 
```

c) Round Robin (pre-emptive)

Aim: To implement Round Robin method of CPU scheduling

Algorithm:

1. Input the list of processes along with their arrival time and burst time.
2. Initialize the time quantum (also known as time slice) for the Round Robin algorithm.
3. Create a queue to hold the processes.
4. Initialize the total waiting time, total turnaround time, and total number of processes.
5. Enqueue all the processes into the queue.
6. While the queue is not empty, do the following:
    a. Dequeue a process from the front of the queue.
    b. Execute the process for the time quantum or until its burst time is completed, whichever is smaller.
    c. Subtract the time quantum or the remaining burst time from the current process.
    d. Update the total waiting time by adding the difference between the current time and the arrival time of the process.

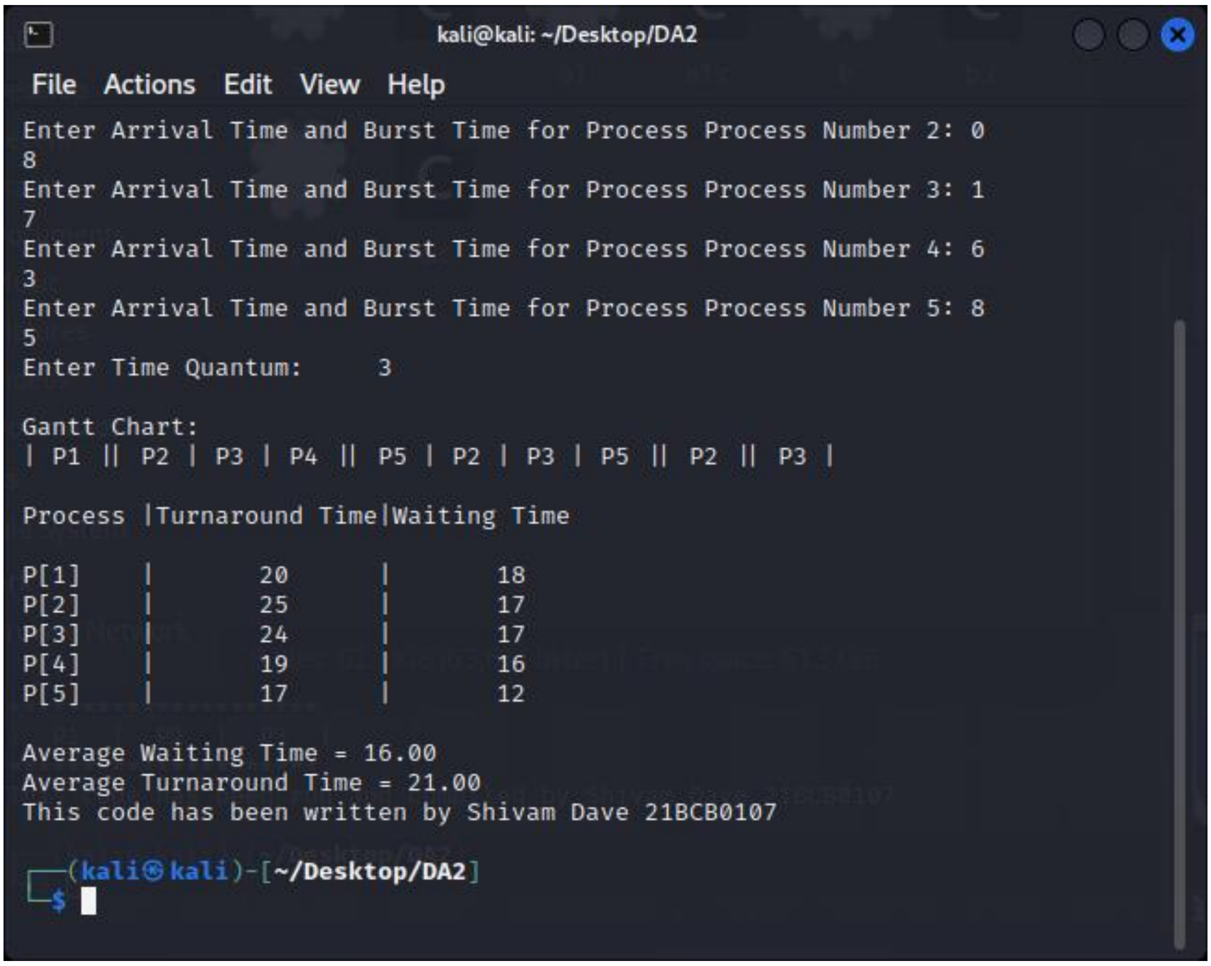
   e. If the process is not completed, enqueue it back into the queue.
   f. If the process is completed, calculate its turnaround time by subtracting its arrival time from the current time, and update the total turnaround time.

7. Calculate the average waiting time by dividing the total waiting time by the number of processes.
8. Calculate the average turnaround time by dividing the total turnaround time by the number of processes.
9. Print the average waiting time and average turnaround time.

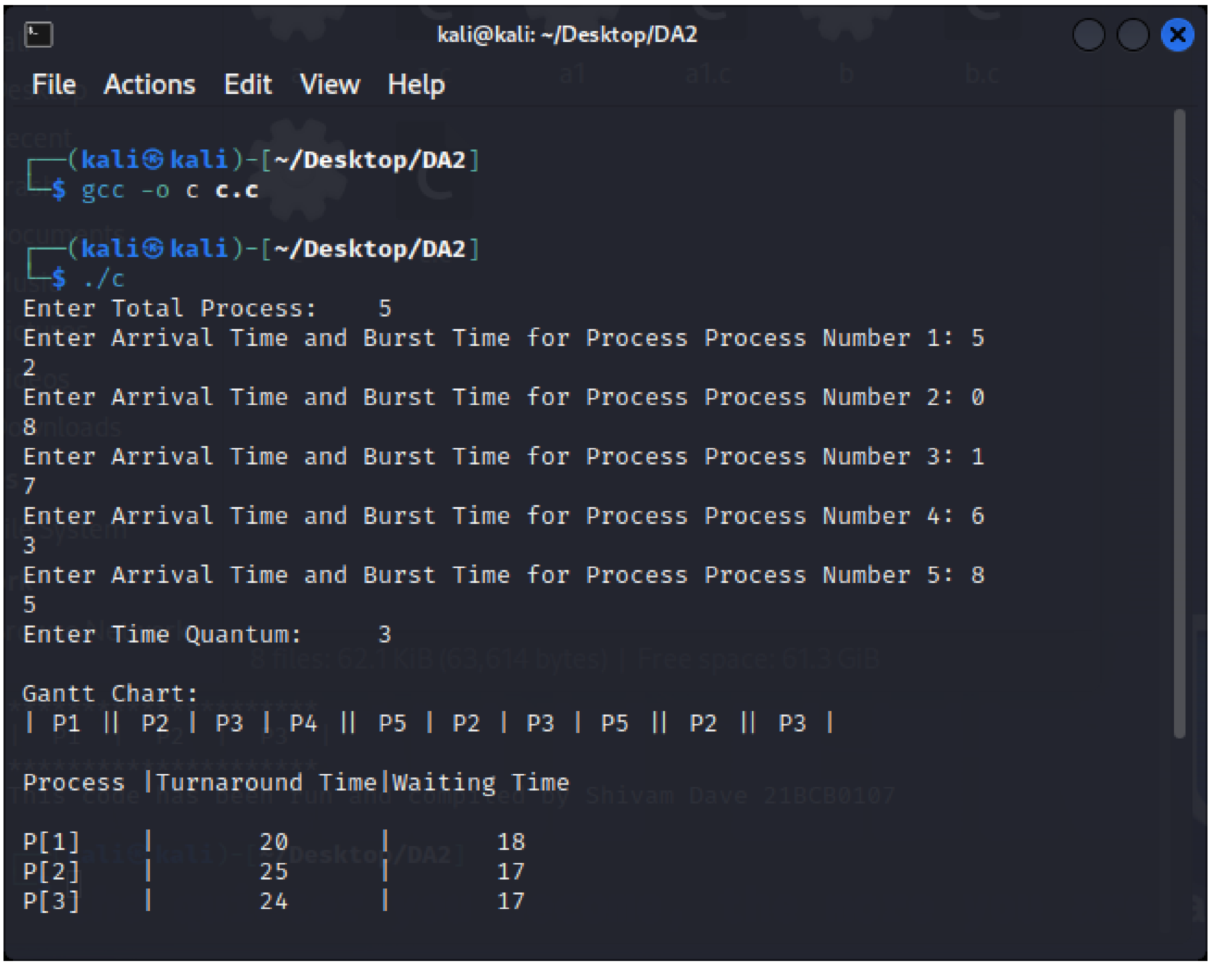
Source Code:

```c
 //ROUND ROBIN JOB SCHEDULING METHOD
#include <stdio.h>
int main()
{

    //Declaring the necessary variables and arrays
    int count, j, n, time, remain, flag = 0, time_quantum;
    int wait_time = 0, turnaround_time = 0, at[10], bt[10], rt[10];

    //Input the no. of processes from user
    printf("Enter Total Process:\t");
    scanf("%d", &n);
    //Initialize remain with n to track the remaining processes
    remain = n;
    /*Using a loop to read & store the arrival time and burst time for each
process from the user.
     Then initializing the remaining time for each process (rt) to be equal to
the burst time.
    */
    for (count = 0; count < n; count++)
    {
        printf("Enter Arrival Time and Burst Time for Process Process Number
%d: ", count + 1);
        scanf("%d", &at[count]);
        scanf("%d", &bt[count]);
        rt[count] = bt[count];
    }
    // Input the time quantum from user
    printf("Enter Time Quantum:\t");
    scanf("%d", &time_quantum);

    // Printing of Gantt chart
    printf("\nGantt Chart:\n");
    int time_slot = 1;
    //Enter a loop where the processes are executed until all processes are
completed i.e. remain is 0
    for (time = 0, count = 0, remain = n; remain != 0;)
    {
    // Check if the remaining time of the current process is less than or
equal to the time quantum and greater than 0.
    if (rt[count] <= time_quantum && rt[count] > 0)
    {
        printf("| P%d ", count + 1);
        time += rt[count];      // Increment the time by the remaining time of
the current process
        rt[count] = 0;          // Set the remaining time of the current
process to 0 as it is completed
        flag = 1;               // Set the flag to indicate that a process has
completed
    }
    // Check if the remaining time of the current process is greater than 0
    else if (rt[count] > 0)
    {
        printf("| P%d ", count + 1);
        rt[count] -= time_quantum;   // Reduce the remaining time of the
current process by the time quantum
        time += time_quantum;        // Increment the time by the time quantum
    }

    // Check if the remaining time of the current process is 0 and the flag is
1 (process completed)
    if (rt[count] == 0 && flag == 1)
    {
        remain--;           // Decrement the count of remaining processes
        printf("|");        // Print a vertical bar to indicate process
completion
        flag = 0;           // Reset the flag
    }

    count++;                // Move to the next process
    if (count == n)
        count = 0;          // Wrap around to the first process if the last
process is reached
    }
    printf("\n");

    //Printing the result
    printf("\nProcess\t|Turnaround Time|Waiting Time\n\n");
    for (count = 0; count < n; count++)
    {
        printf("P[%d]\t|\t%d\t|\t%d\n", count + 1, time - at[count], time -
at[count] - bt[count]);
        wait_time += time - at[count] - bt[count];
        turnaround_time += time - at[count];
    }
    printf("\nAverage Waiting Time = %.2f\n", wait_time * 1.0 / n);
    printf("Average Turnaround Time = %.2f\n", turnaround_time * 1.0 / n);
    printf("This code has been written by Shivam Dave 21BCB0107");
    return 0;
}
```

Result and Output:



```
Enter Arrival Time and Burst Time for Process Process Number 2: 0
8
Enter Arrival Time and Burst Time for Process Process Number 3: 1
7
Enter Arrival Time and Burst Time for Process Process Number 4: 6
3
Enter Arrival Time and Burst Time for Process Process Number 5: 8
5
Enter Time Quantum:     3

Gantt Chart:
| P1 || P2 | P3 | P4 || P5 | P2 | P3 | P5 || P2 || P3 |

Process |Turnaround Time|Waiting Time

P[1]    |       20      |       18
P[2]    |       25      |       17
P[3]    |       24      |       17
P[4]    |       19      |       16
P[5]    |       17      |       12

Average Waiting Time = 16.00
Average Turnaround Time = 21.00
This code has been written by Shivam Dave 21BCB0107

┌──(kali㉿kali)-[~/Desktop/DA2]
└─$ 
```



```
┌──(kali㉿kali)-[~/Desktop/DA2]
└─$ gcc -o c c.c

┌──(kali㉿kali)-[~/Desktop/DA2]
└─$ ./c
Enter Total Process:    5
Enter Arrival Time and Burst Time for Process Process Number 1: 5
2
Enter Arrival Time and Burst Time for Process Process Number 2: 0
8
Enter Arrival Time and Burst Time for Process Process Number 3: 1
7
Enter Arrival Time and Burst Time for Process Process Number 4: 6
3
Enter Arrival Time and Burst Time for Process Process Number 5: 8
5
Enter Time Quantum:     3

Gantt Chart:
| P1 || P2 | P3 | P4 || P5 | P2 | P3 | P5 || P2 || P3 |

Process |Turnaround Time|Waiting Time

P[1]    |       20      |       18
P[2]    |       25      |       17
P[3]    |       24      |       17
```

d) Priority

Aim: To implement priority algorithm for CPU scheduling

Algorithm:

1. Input the list of processes along with their arrival time, burst time, and priority.
2. Create a priority queue to hold the processes, where the priority is the key for ordering the processes.
3. Initialize the total waiting time, total turnaround time, and total number of processes.
4. Enqueue all the processes into the priority queue.
5. While the priority queue is not empty, do the following:
    a. Dequeue the process with the highest priority from the front of the priority queue.
    b. Execute the process until its burst time is completed.
    c. Subtract the burst time of the current process by the time it was executed.

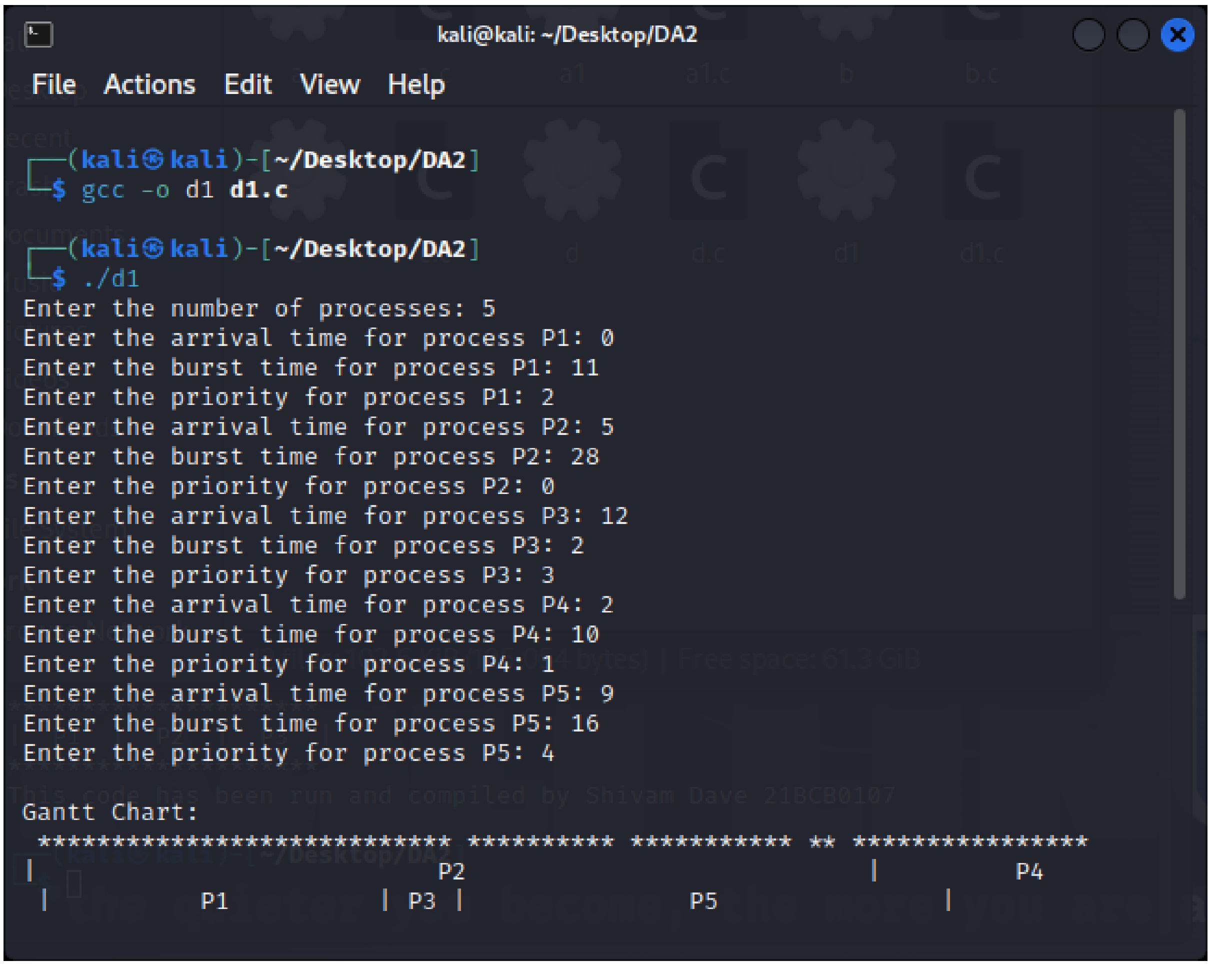
   d. Update the total waiting time by adding the difference between the current time and the arrival time of the process.
   e. If the process is not completed, enqueue it back into the priority queue with its updated priority.
   f. If the process is completed, calculate its turnaround time by subtracting its arrival time from the current time, and update the total turnaround time.
6. Calculate the average waiting time by dividing the total waiting time by the number of processes.
7. Calculate the average turnaround time by dividing the total turnaround time by the number of processes.
8. Print the average waiting time and average turnaround time.

Source Code:

```c
//Priority Scheduling
#include <stdio.h>
// Creating a structure to represent a process
struct Process {
    int processID;        // Process ID
    int arrivalTime;      // Arrival time of the process
    int burstTime;        // Burst time of the process
    int priority;         // Priority of the process
};

// Creating a function to calculate average waiting time and average
turnaround time
void calculateAverageTimes(struct Process processes[], int n) {
    int i, j, totalWaitingTime = 0, totalTurnaroundTime = 0;

    // Calculating the waiting time and turnaround time for each process
    int waitingTime[n], turnaroundTime[n];
    waitingTime[0] = 0; // Waiting time for the first process is always 0

    for (i = 1; i < n; i++)
    {
        waitingTime[i] = processes[i - 1].burstTime + waitingTime[i - 1];
        /*
        Loop to calculates waiting time for each process starting from the
second process (i = 1).
        It is calculated by adding the burst time of the previous process and
the waiting time of
        the previous process. And store the waiting time in its array
        */
    }
    for (i = 0; i < n; i++)
    {
        /*
        loop to calculates turnaround time for each process and updates the
totalWaitingTime
        and totalTurnaroundTime variables.It is calculated by adding the burst
time and waiting
        time of each process.
        */
        turnaroundTime[i] = processes[i].burstTime + waitingTime[i];
        totalWaitingTime += waitingTime[i];
        totalTurnaroundTime += turnaroundTime[i];
    }

    // Printing of the Gantt chart
    printf("\nGantt Chart:\n");
    printf(" ");
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < processes[i].burstTime; j++)
        {
            printf("*");
        }
        printf(" ");
    }
    printf("\n|");
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < processes[i].burstTime - 1; j++)
        {
            printf(" ");
        }
        printf("P%d", processes[i].processID);
        for (j = 0; j < processes[i].burstTime - 1; j++)
        {
            printf(" ");
        }
        printf("|");
    }
    printf("\n ");
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < processes[i].burstTime; j++)
        {
            printf("*");
        }
        printf(" ");
    }
    printf("\n");
    printf("0");
    for (i = 0; i < n; i++) {
        for (j = 0; j < processes[i].burstTime; j++) {
            printf("  ");
        }
        printf("%d", turnaroundTime[i]);
    }
    printf("\n");


    // Printing the process details
    printf("\nProcess Details:\n");
    printf("Process\tArrival Time\tBurst Time\tPriority\tWaiting
Time\tTurnaround Time\n");
    for (i = 0; i < n; i++)
    {
        printf("P%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", processes[i].processID,
processes[i].arrivalTime, processes[i].burstTime, processes[i].priority,
waitingTime[i], turnaroundTime[i]);
    }
    // Calculate the average waiting time and average turnaround time by
dividing it by n
    float avgWaitingTime = (float)totalWaitingTime / n;
    float avgTurnaroundTime = (float)totalTurnaroundTime / n;

    printf("\nAverage Waiting Time: %.2f\n", avgWaitingTime);
    printf("Average Turnaround Time: %.2f\n", avgTurnaroundTime);
}
int main()
{
    int n, i;

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

    // Create an array of processes
    struct Process processes[n];

    // Get the arrival time, burst time, and priority for each process from
the user
    for (i = 0; i < n; i++)
    {
        printf("Enter the arrival time for process P%d: ", i + 1);
        scanf("%d", &processes[i].arrivalTime);
        printf("Enter the burst time for process P%d: ", i + 1);
        scanf("%d", &processes[i].burstTime);
        printf("Enter the priority for process P%d: ", i + 1);
        scanf("%d", &processes[i].priority);
        processes[i].processID = i + 1;
    }
    // Sort the processes based on priority in ascending order
    for (i = 0; i < n - 1; i++)
    {
        for (int j = 0; j < n - i - 1; j++)
        {
            if (processes[j].priority > processes[j + 1].priority)
            {
                // Swap the processes
                struct Process temp = processes[j];
                processes[j] = processes[j + 1];
                processes[j + 1] = temp;
            }
        }
    }

    // Calculate average waiting time and average turnaround time
    calculateAverageTimes(processes, n);
    printf("This code has been written by Shivam Dave 21BCB0107");
    return 0;
}
```

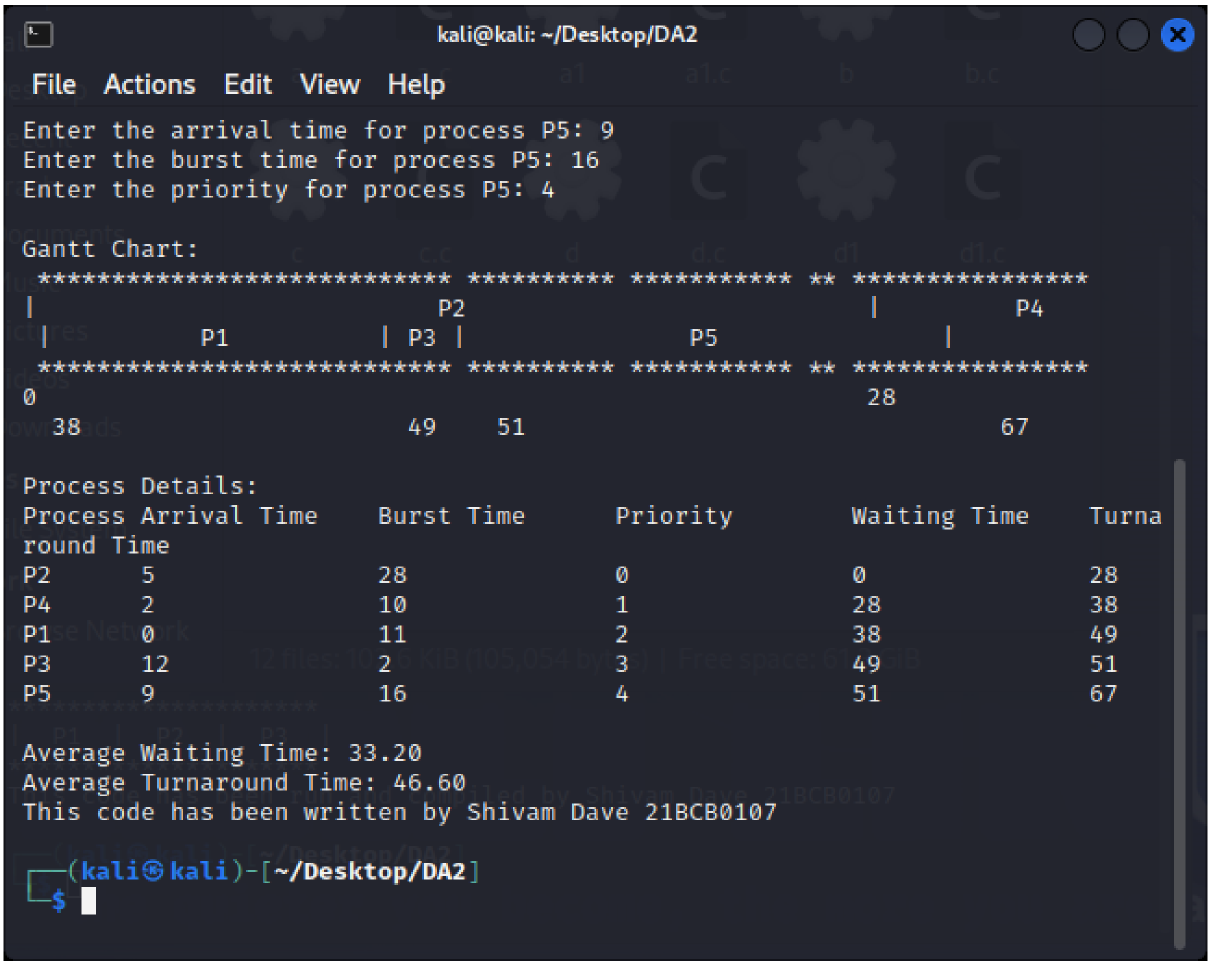
Result and Output:

21BCB0107



```
┌──(kali㉿kali)-[~/Desktop/DA2]
└─$ gcc -o d1 d1.c

┌──(kali㉿kali)-[~/Desktop/DA2]
└─$ ./d1
Enter the number of processes: 5
Enter the arrival time for process P1: 0
Enter the burst time for process P1: 11
Enter the priority for process P1: 2
Enter the arrival time for process P2: 5
Enter the burst time for process P2: 28
Enter the priority for process P2: 0
Enter the arrival time for process P3: 12
Enter the burst time for process P3: 2
Enter the priority for process P3: 3
Enter the arrival time for process P4: 2
Enter the burst time for process P4: 10
Enter the priority for process P4: 1
Enter the arrival time for process P5: 9
Enter the burst time for process P5: 16
Enter the priority for process P5: 4

Gantt Chart:
 **************************** ********** *********** ** ****************
|                            P2                            |          P4
 |           P1           | P3 |                  P5                 |
```

21BCB0107


```
Enter the arrival time for process P5: 9
Enter the burst time for process P5: 16
Enter the priority for process P5: 4

Gantt Chart:
 **************************** ********** *********** ** ****************
|                            P2                            |         P4
 |          P1          | P3 |               P5                |
 **************************** ********** *********** ** ****************
0                                                        28
  38                      49     51                                 67

Process Details:
Process Arrival Time    Burst Time      Priority        Waiting Time    Turna
round Time
P2      5               28              0               0               28
P4      2               10              1               28              38
P1      0               11              2               38              49
P3      12              2               3               49              51
P5      9               16              4               51              67

Average Waiting Time: 33.20
Average Turnaround Time: 46.60
This code has been written by Shivam Dave 21BCB0107

┌──(kali㉿kali)-[~/Desktop/DA2]
└─$ 
```