Name : Kishaanth.S

Reg.No : 205001054

Exercise – 4

Implementation of CPU Scheduling Policies: Priority (Pre-emptive) and Round Robin

Aim :

To develop a menu driven c program to implement the CPU scheduling algorithms, Priority and Round Robin.

Algorithm : (Priority-preemptive)

```
1: Input the number of processes from the user.
2: Have a structure with pid, waiting, burst, arrival, turn_around and completion as data members.
3: Using a loop, input all the given details for each process and store it.
4: Store the burst time of each process in array names rem_time.
5: Have an additional array of type of that structure.
6: Create and initialize ptr to 0,completed to 0 and cur_time to 0.
7. While all the processes are not completed,
    7.1: Initialize index to -1.
    7.2: Using a loop for all the processes, check which has the highest priority and store it in index.
    7.3: Increment cur_time by 1.
    7.4: If ptr is not 0 and id of previous process is same as that of the current selected process, decrement ptr by 1.
    7.5: Else store id of that process to the additional created array and assign burst of that process to 0.
    7.6: Increment the burst time of that process by 1 and also assign the completion time of that process as cur_time.
    7.7: Increment ptr by 1 and decrement the rem_time of that process by 1.
    7.8: If rem_time of that process is 0,
        7.8.1: increment completed by 1.
```

```
            7.8.2: completion time of that process is the sum of the burst time and arrival time of that process.
            7.8.3: Waiting time of current process is equal to the difference between the completion time and sum of arrival and burst of that current process.
            7.8.4: average weight is incremented by the waiting time of that process.
            7.8.5 : turn_around of that process is the difference between the completion time and arrival time of that process.
            7.8.6 : average_ta is incremented by the turn_around of that process.

    8: Print all the details of the processes. Also print the gantt chart along with the average weighting time and average turn_around time.
```

Algorithm : (Round Robin)

```
    1: Input the number of processes from the user.
    2: Have a structure with pid, waiting, burst, arrival, turn_around and completion as data members.
    3: Using a loop, input all the given details for each process and store it.
    4: Store the burst time of each process in array names rem_time.
    5: Have an additional array of type of that structure.
    6: Create and initialize ptr to 0,completed to 0 and cur_time to 0.
    7. While all the processes are not completed,
        7.1: for all process,
            7.1.1: if rem_time of that process is greater than 0, assign process to ptr.
                7.1.1.1: If rem_time of current process is greater than the quantum, increment cur_time by quantum and also store decrement rem_time by quantum.
                7.1.1.2: Else burst time of current process is rem_time of that process. Increment cur_time by rem_time of that process.
                7.1.1.3: : completion time of that process is the sum of the burst time and arrival time of that process.
                7.1.1.4: Waiting time of current process is equal to the difference between the completion time and sum of arrival and burst of that current process.
                7.1.1.5: average weight is incremented by the waiting time of that process.
```

7.1.1.6 : turn_around of that process is the difference between the completion time and arrival time of that process.

7.1.1.7 : average_ta is incremented by the turn_around of that process.

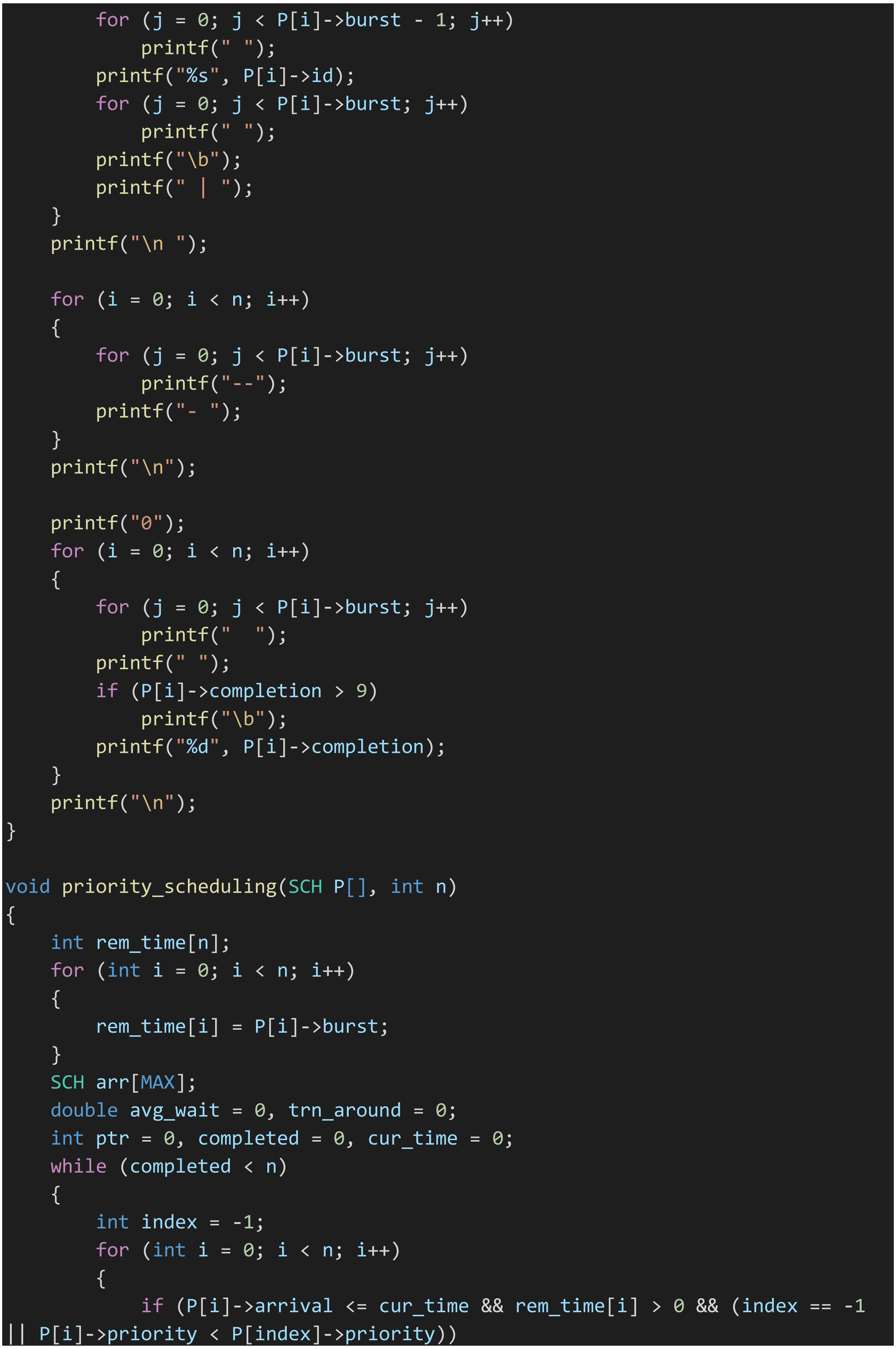
7.1.1.8: Increment completed by 1.

7.1.2: Completion time of current process is equal to the cur_time and increment ptr by 1.

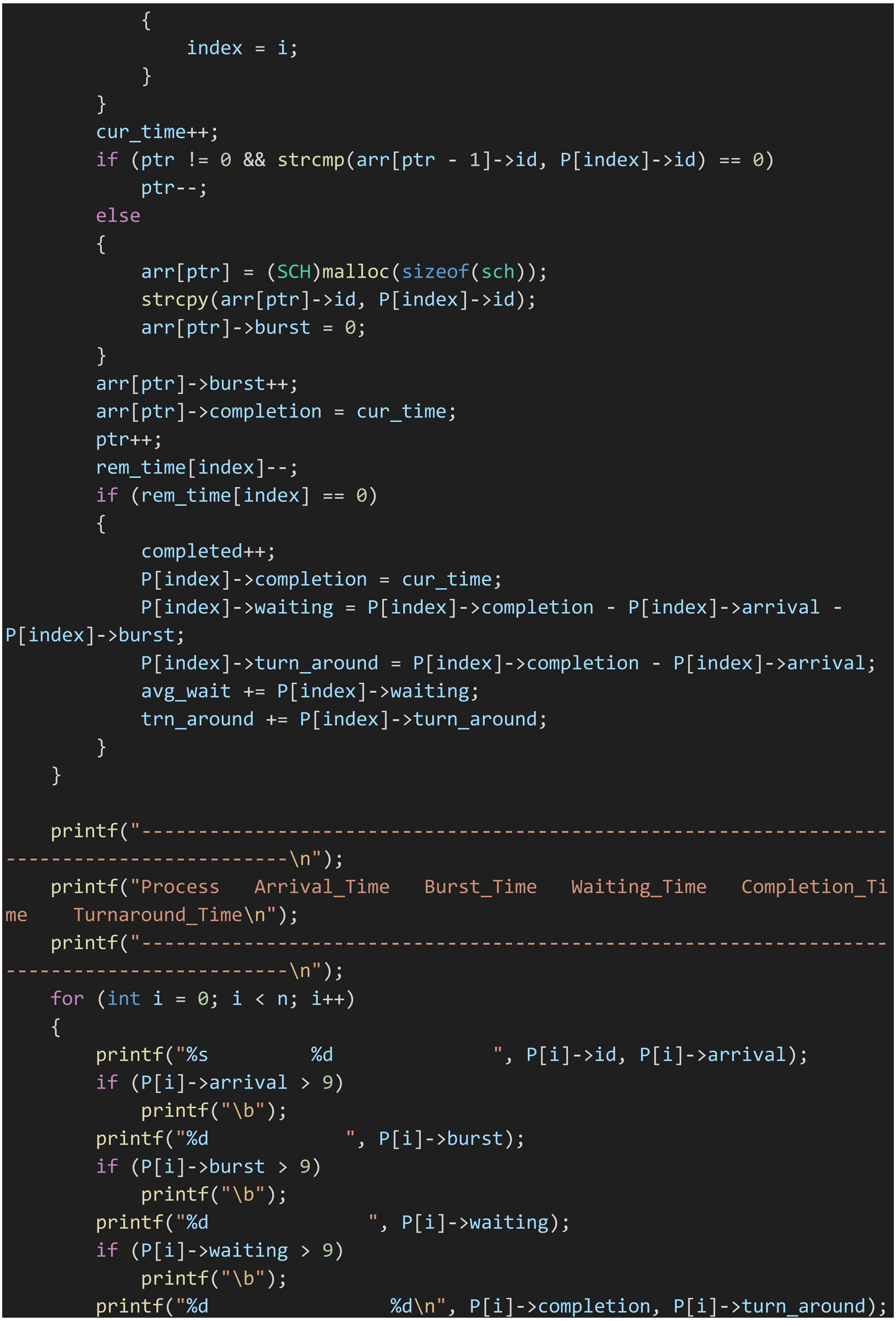
8: Print all the details of the processes. Also print the gantt chart along with the average weighting time and average turn_around time.

Source Code :

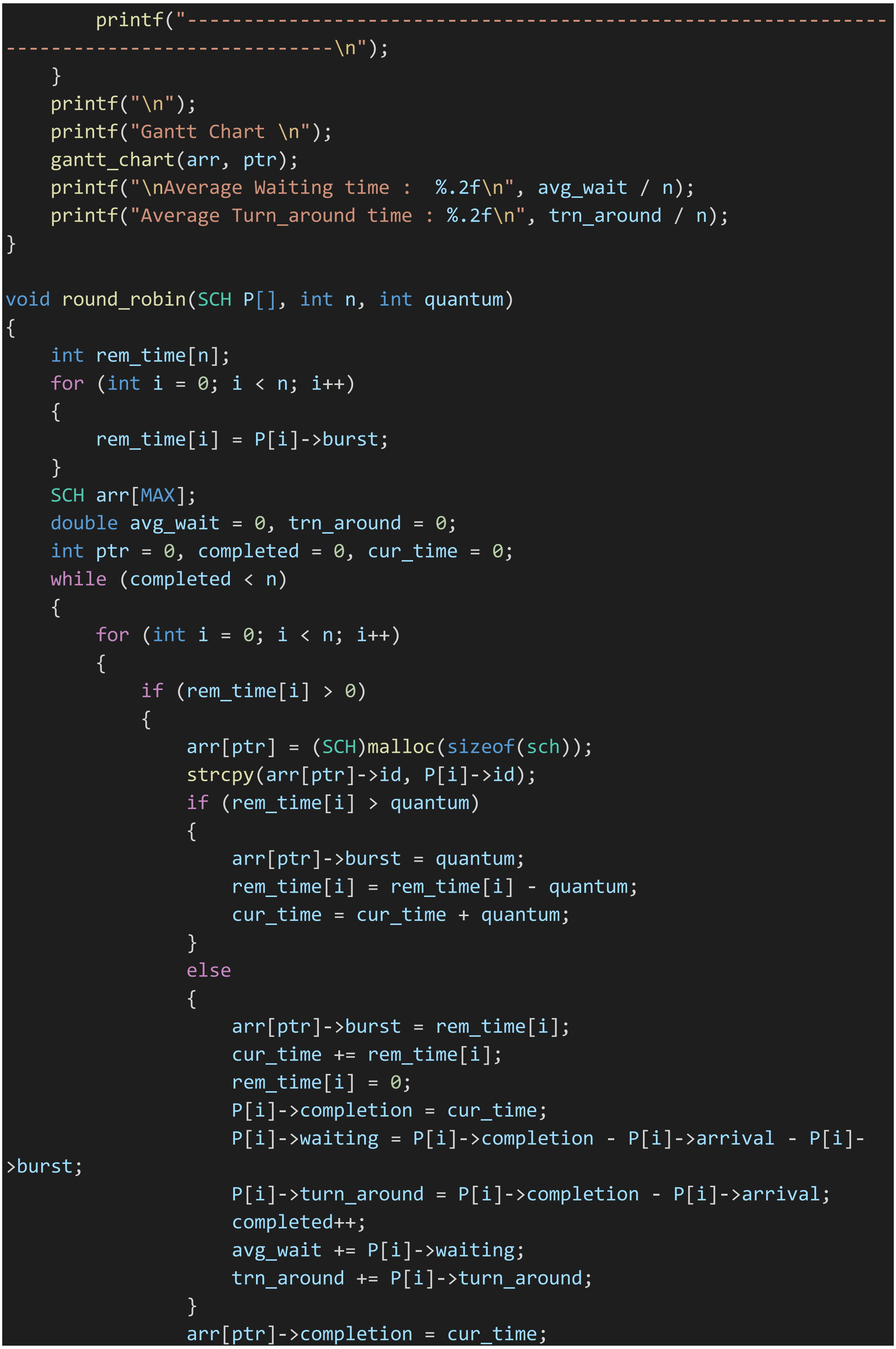
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX 100
typedef struct schedule *SCH;
typedef struct schedule
{
    char id[3];
    int waiting;
    int arrival;
    int turn_around;
    int burst;
    int completion;
    int priority;
} sch;

void gantt_chart(SCH P[], int n)
{
    int i, j;

    printf(" ");
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < P[i]->burst; j++)
            printf("--");

        printf("- ");
    }
    printf("\n| ");

    for (i = 0; i < n; i++)
    {
```



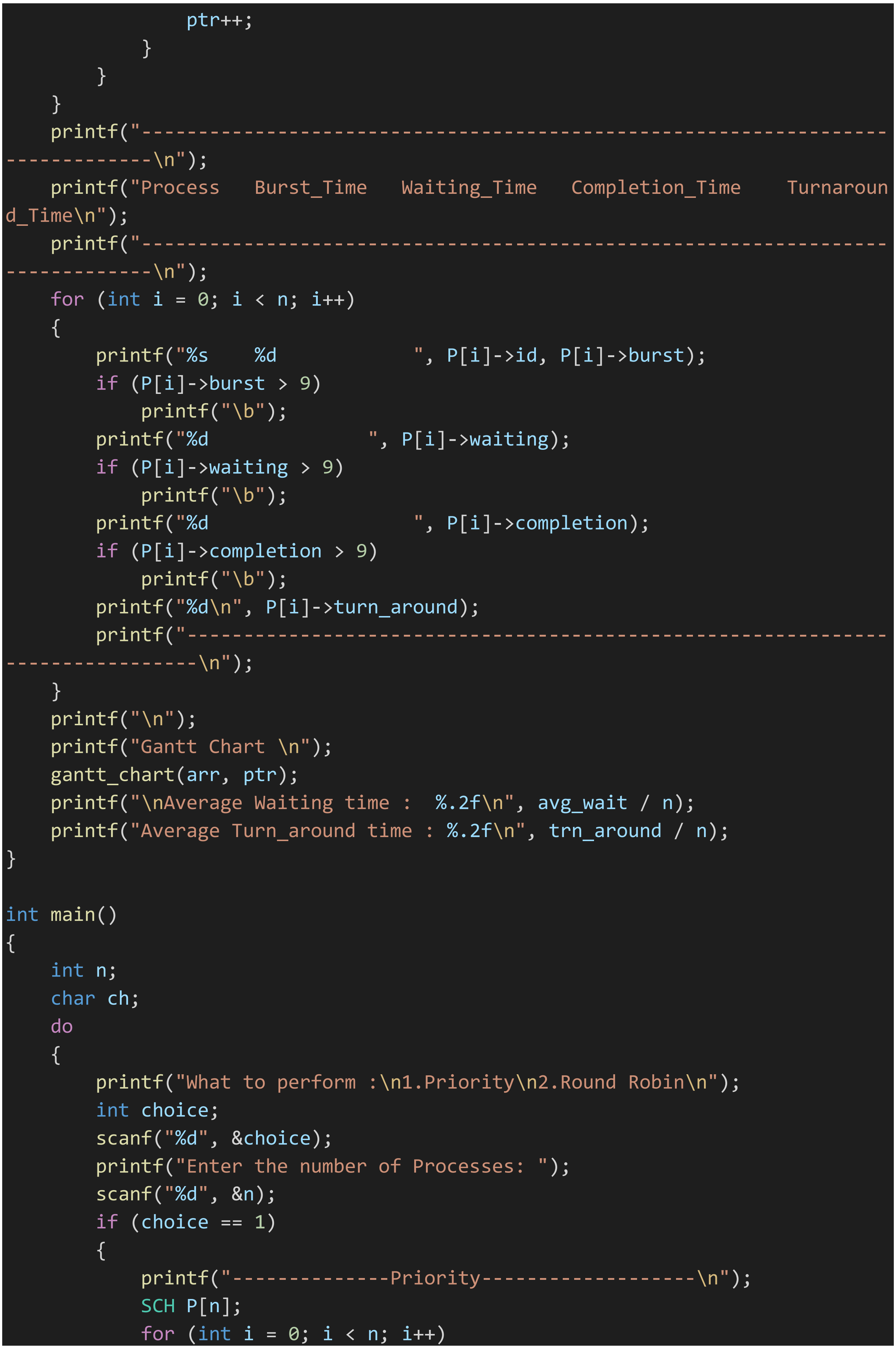
```
        for (j = 0; j < P[i]->burst - 1; j++)
            printf(" ");
        printf("%s", P[i]->id);
        for (j = 0; j < P[i]->burst; j++)
            printf(" ");
        printf("\b");
        printf(" | ");
    }
    printf("\n ");

    for (i = 0; i < n; i++)
    {
        for (j = 0; j < P[i]->burst; j++)
            printf("--");
        printf("- ");
    }
    printf("\n");

    printf("0");
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < P[i]->burst; j++)
            printf("  ");
        printf(" ");
        if (P[i]->completion > 9)
            printf("\b");
        printf("%d", P[i]->completion);
    }
    printf("\n");
}

void priority_scheduling(SCH P[], int n)
{
    int rem_time[n];
    for (int i = 0; i < n; i++)
    {
        rem_time[i] = P[i]->burst;
    }
    SCH arr[MAX];
    double avg_wait = 0, trn_around = 0;
    int ptr = 0, completed = 0, cur_time = 0;
    while (completed < n)
    {
        int index = -1;
        for (int i = 0; i < n; i++)
        {
            if (P[i]->arrival <= cur_time && rem_time[i] > 0 && (index == -1
|| P[i]->priority < P[index]->priority))
```



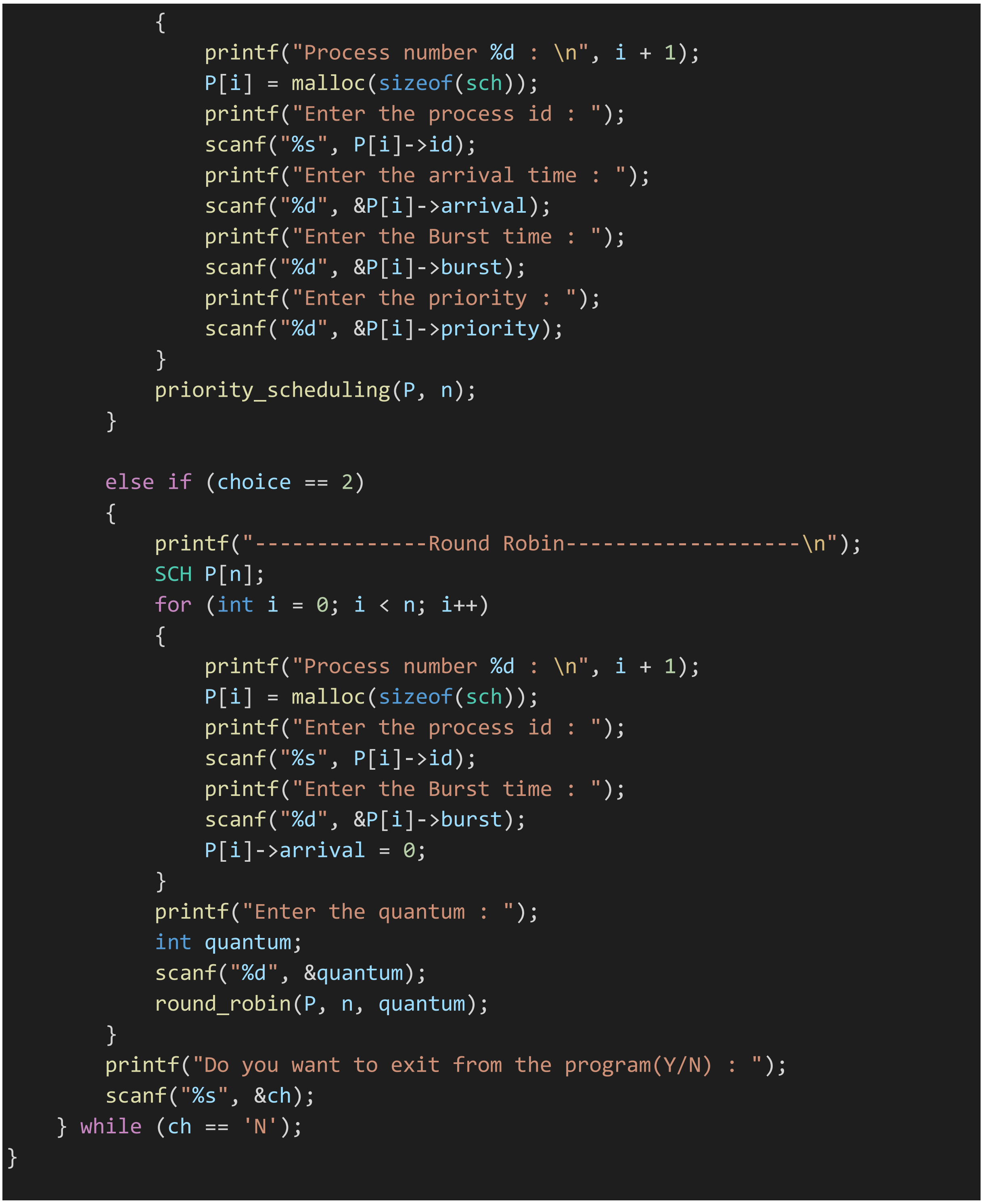
```
            {
                index = i;
            }
        }
        cur_time++;
        if (ptr != 0 && strcmp(arr[ptr - 1]->id, P[index]->id) == 0)
            ptr--;
        else
        {
            arr[ptr] = (SCH)malloc(sizeof(sch));
            strcpy(arr[ptr]->id, P[index]->id);
            arr[ptr]->burst = 0;
        }
        arr[ptr]->burst++;
        arr[ptr]->completion = cur_time;
        ptr++;
        rem_time[index]--;
        if (rem_time[index] == 0)
        {
            completed++;
            P[index]->completion = cur_time;
            P[index]->waiting = P[index]->completion - P[index]->arrival - 
P[index]->burst;
            P[index]->turn_around = P[index]->completion - P[index]->arrival;
            avg_wait += P[index]->waiting;
            trn_around += P[index]->turn_around;
        }
    }

    printf("------------------------------------------------------------------
-------------------------\n");
    printf("Process   Arrival_Time   Burst_Time   Waiting_Time   Completion_Ti
me    Turnaround_Time\n");
    printf("------------------------------------------------------------------
-------------------------\n");
    for (int i = 0; i < n; i++)
    {
        printf("%s         %d              ", P[i]->id, P[i]->arrival);
        if (P[i]->arrival > 9)
            printf("\b");
        printf("%d            ", P[i]->burst);
        if (P[i]->burst > 9)
            printf("\b");
        printf("%d              ", P[i]->waiting);
        if (P[i]->waiting > 9)
            printf("\b");
        printf("%d                %d\n", P[i]->completion, P[i]->turn_around);
```



```
        printf("--------------------------------------------------------------
-----------------------------\n");
    }
    printf("\n");
    printf("Gantt Chart \n");
    gantt_chart(arr, ptr);
    printf("\nAverage Waiting time :  %.2f\n", avg_wait / n);
    printf("Average Turn_around time : %.2f\n", trn_around / n);
}

void round_robin(SCH P[], int n, int quantum)
{
    int rem_time[n];
    for (int i = 0; i < n; i++)
    {
        rem_time[i] = P[i]->burst;
    }
    SCH arr[MAX];
    double avg_wait = 0, trn_around = 0;
    int ptr = 0, completed = 0, cur_time = 0;
    while (completed < n)
    {
        for (int i = 0; i < n; i++)
        {
            if (rem_time[i] > 0)
            {
                arr[ptr] = (SCH)malloc(sizeof(sch));
                strcpy(arr[ptr]->id, P[i]->id);
                if (rem_time[i] > quantum)
                {
                    arr[ptr]->burst = quantum;
                    rem_time[i] = rem_time[i] - quantum;
                    cur_time = cur_time + quantum;
                }
                else
                {
                    arr[ptr]->burst = rem_time[i];
                    cur_time += rem_time[i];
                    rem_time[i] = 0;
                    P[i]->completion = cur_time;
                    P[i]->waiting = P[i]->completion - P[i]->arrival - P[i]-
>burst;

                    P[i]->turn_around = P[i]->completion - P[i]->arrival;
                    completed++;
                    avg_wait += P[i]->waiting;
                    trn_around += P[i]->turn_around;
                }
                arr[ptr]->completion = cur_time;
```

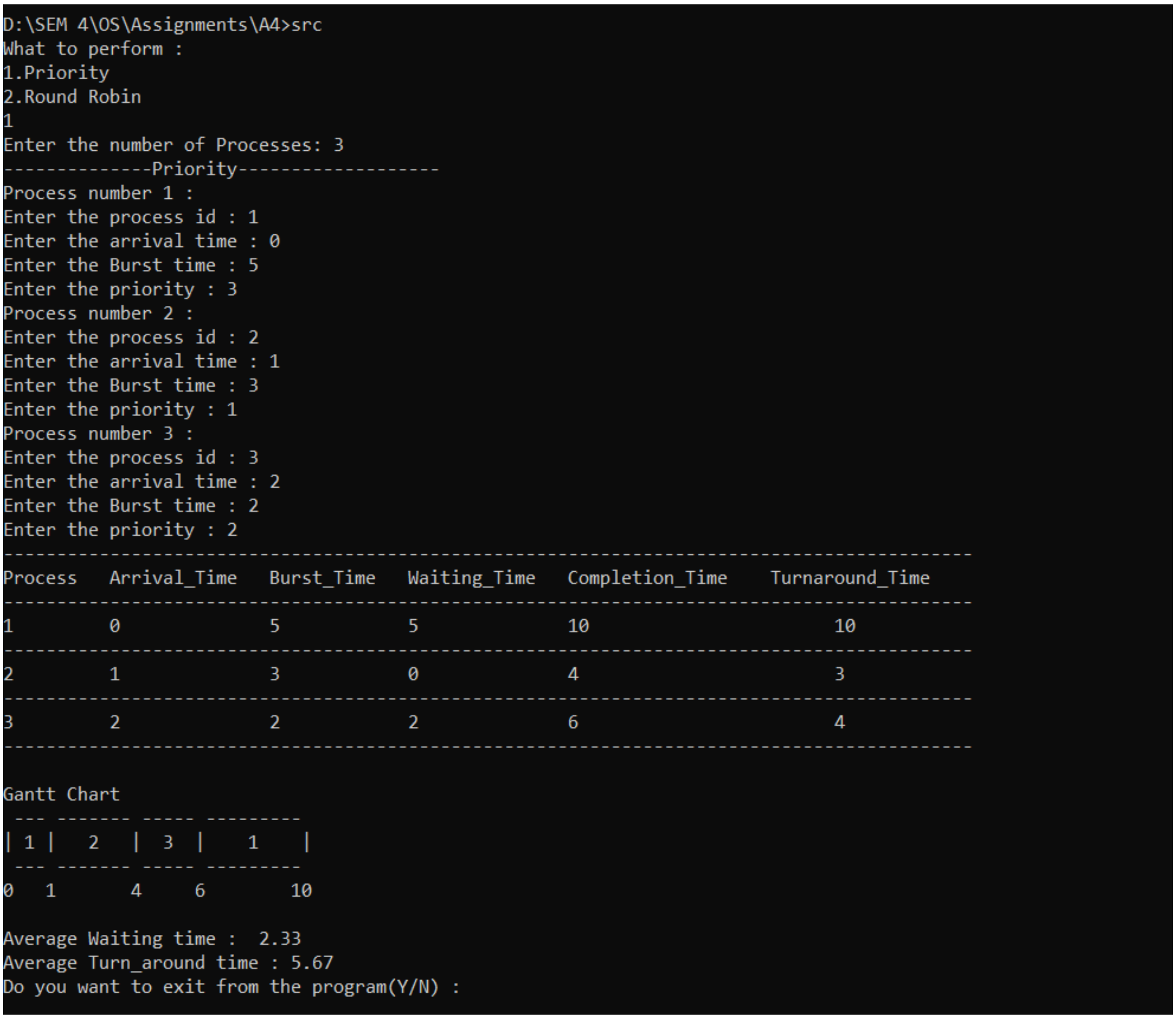


```
                ptr++;
            }
        }
    }
    printf("------------------------------------------------------------------
-------------\n");
    printf("Process   Burst_Time   Waiting_Time   Completion_Time    Turnaroun
d_Time\n");
    printf("------------------------------------------------------------------
-------------\n");
    for (int i = 0; i < n; i++)
    {
        printf("%s    %d            ", P[i]->id, P[i]->burst);
        if (P[i]->burst > 9)
            printf("\b");
        printf("%d              ", P[i]->waiting);
        if (P[i]->waiting > 9)
            printf("\b");
        printf("%d                  ", P[i]->completion);
        if (P[i]->completion > 9)
            printf("\b");
        printf("%d\n", P[i]->turn_around);
        printf("--------------------------------------------------------------
-----------------\n");
    }
    printf("\n");
    printf("Gantt Chart \n");
    gantt_chart(arr, ptr);
    printf("\nAverage Waiting time :  %.2f\n", avg_wait / n);
    printf("Average Turn_around time : %.2f\n", trn_around / n);
}

int main()
{
    int n;
    char ch;
    do
    {
        printf("What to perform :\n1.Priority\n2.Round Robin\n");
        int choice;
        scanf("%d", &choice);
        printf("Enter the number of Processes: ");
        scanf("%d", &n);
        if (choice == 1)
        {
            printf("--------------Priority-------------------\n");
            SCH P[n];
            for (int i = 0; i < n; i++)
```

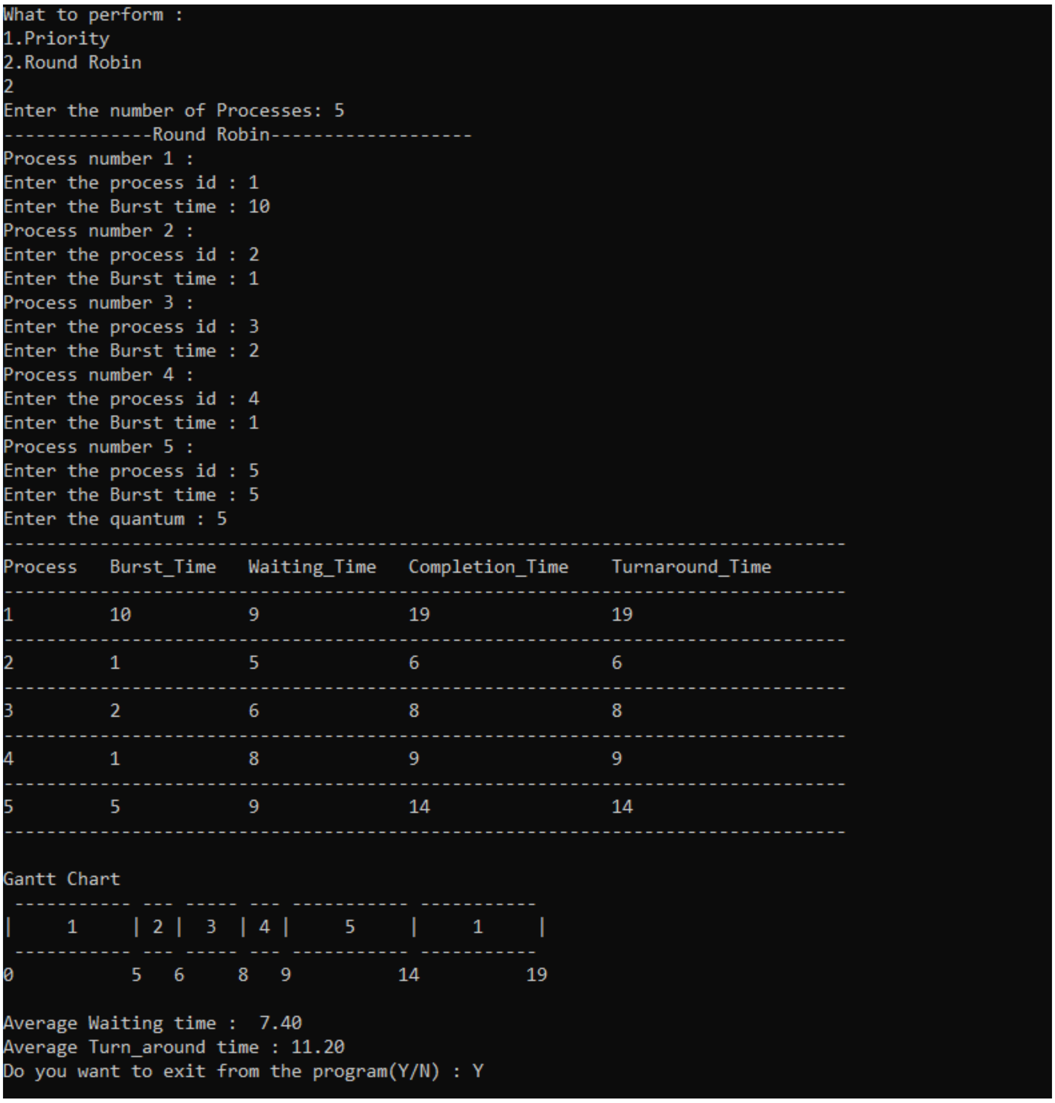


```
            {
                printf("Process number %d : \n", i + 1);
                P[i] = malloc(sizeof(sch));
                printf("Enter the process id : ");
                scanf("%s", P[i]->id);
                printf("Enter the arrival time : ");
                scanf("%d", &P[i]->arrival);
                printf("Enter the Burst time : ");
                scanf("%d", &P[i]->burst);
                printf("Enter the priority : ");
                scanf("%d", &P[i]->priority);
            }
            priority_scheduling(P, n);
        }

        else if (choice == 2)
        {
            printf("--------------Round Robin-------------------\n");
            SCH P[n];
            for (int i = 0; i < n; i++)
            {
                printf("Process number %d : \n", i + 1);
                P[i] = malloc(sizeof(sch));
                printf("Enter the process id : ");
                scanf("%s", P[i]->id);
                printf("Enter the Burst time : ");
                scanf("%d", &P[i]->burst);
                P[i]->arrival = 0;
            }
            printf("Enter the quantum : ");
            int quantum;
            scanf("%d", &quantum);
            round_robin(P, n, quantum);
        }
        printf("Do you want to exit from the program(Y/N) : ");
        scanf("%s", &ch);
    } while (ch == 'N');
}
```

Output :



```
D:\SEM 4\OS\Assignments\A4>src
What to perform :
1.Priority
2.Round Robin
1
Enter the number of Processes: 3
--------------Priority------------------
Process number 1 :
Enter the process id : 1
Enter the arrival time : 0
Enter the Burst time : 5
Enter the priority : 3
Process number 2 :
Enter the process id : 2
Enter the arrival time : 1
Enter the Burst time : 3
Enter the priority : 1
Process number 3 :
Enter the process id : 3
Enter the arrival time : 2
Enter the Burst time : 2
Enter the priority : 2
-------------------------------------------------------------------------------------------
Process   Arrival_Time   Burst_Time   Waiting_Time   Completion_Time    Turnaround_Time
-------------------------------------------------------------------------------------------
1         0              5            5              10                       10
-------------------------------------------------------------------------------------------
2         1              3            0              4                        3
-------------------------------------------------------------------------------------------
3         2              2            2              6                        4
-------------------------------------------------------------------------------------------

Gantt Chart
 --- ------- ----- ---------
| 1 |   2   |  3  |    1    |
 --- ------- ----- ---------
0   1       4     6        10

Average Waiting time :  2.33
Average Turn_around time : 5.67
Do you want to exit from the program(Y/N) :
```



```
What to perform :
1.Priority
2.Round Robin
2
Enter the number of Processes: 5
--------------Round Robin-------------------
Process number 1 :
Enter the process id : 1
Enter the Burst time : 10
Process number 2 :
Enter the process id : 2
Enter the Burst time : 1
Process number 3 :
Enter the process id : 3
Enter the Burst time : 2
Process number 4 :
Enter the process id : 4
Enter the Burst time : 1
Process number 5 :
Enter the process id : 5
Enter the Burst time : 5
Enter the quantum : 5
-------------------------------------------------------------------------------
Process   Burst_Time   Waiting_Time   Completion_Time    Turnaround_Time
-------------------------------------------------------------------------------
1         10           9              19                 19
-------------------------------------------------------------------------------
2         1            5              6                  6
-------------------------------------------------------------------------------
3         2            6              8                  8
-------------------------------------------------------------------------------
4         1            8              9                  9
-------------------------------------------------------------------------------
5         5            9              14                 14
-------------------------------------------------------------------------------

Gantt Chart
 ----------- --- ----- --- ----------- -----------
|     1     | 2 |  3  | 4 |     5     |     1     |
 ----------- --- ----- --- ----------- -----------
0           5   6     8   9          14          19

Average Waiting time :  7.40
Average Turn_around time : 11.20
Do you want to exit from the program(Y/N) : Y
```