Program

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

typedef struct Process
{
    int id;
    int arrival_time;
    int burst_time;
    int remaining_time;
    int waiting_time;
    int turnaround_time;
    int priority;
} Process;

double calculate_avg_waiting_time(Process processes[], int n)
{
    double total_waiting_time = 0;
    for (int i = 0; i < n; i++)
    {
        total_waiting_time += processes[i].waiting_time;
    }
    return total_waiting_time / n;
}

void FCFS(Process processes[], int n)
{
    int time = 0;
    for (int i = 0; i < n - 1; i++)
    {
        for (int j = i + 1; j < n; j++)
        {
            if (processes[i].arrival_time > processes[j].arrival_time)
            {
                Process temp = processes[i];
                processes[i] = processes[j];
                processes[j] = temp;
            }
        }
    }
    for (int i = 0; i < n; i++)
    {
        if (time < processes[i].arrival_time)
        {
            time = processes[i].arrival_time;
```

```
        }
        printf("%d|%d|", time, processes[i].id);
        processes[i].waiting_time = time - processes[i].arrival_time;
        time += processes[i].burst_time;
        processes[i].turnaround_time = processes[i].waiting_time + processes[i].burst_time;
         printf("%d ---> ", time);
    }
}

void SJF(Process processes[], int n)
{
    int time = 0;
    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 || (processes[i].burst_time == pro
            {
             Process temp = processes[i];
             processes[i] = processes[j];
             processes[j] = temp;
            }
        }
    }
    for (int i = 0; i < n; i++)
    {
        if (time < processes[i].arrival_time)
        {
            time = processes[i].arrival_time;
        }
        printf("%d|%d|", time, processes[i].id);
        processes[i].waiting_time = time - processes[i].arrival_time;
        time += processes[i].burst_time;
        processes[i].turnaround_time = processes[i].waiting_time + processes[i].burst_time;
         printf("%d ---> ", time);
    }
}

void SRTF(Process processes[], int n)
{
    int time = 0, completed = 0;
    int remaining_burst_times[n];
    for (int i = 0; i < n; i++)
    {
        remaining_burst_times[i] = processes[i].burst_time;
    }
```

```
    while (completed < n)
    {
 int shortest = -1;
for (int i = 0; i < n; i++)
{
if (processes[i].arrival_time <= time && remaining_burst_times[i] > 0 && (shortest == -1 ||
            {
                shortest = i;
            }
        }
        if (shortest == -1)
        {
            time++;
            continue;
        }
        printf("%d|%d|", time, processes[shortest].id);
        remaining_burst_times[shortest]--;
        if (remaining_burst_times[shortest] == 0)
        {
            completed++;
processes[shortest].waiting_time = time + 1 - processes[shortest].arrival_time - processes[s
processes[shortest].turnaround_time = processes[shortest].waiting_time + processes[shortest]
        }
        time++;
        printf("%d ---> ", time);
    }
}

void priority_scheduling(Process processes[], int n)
{
    int time = 0;
    for (int i = 0; i < n - 1; i++)
    {
        for (int j = i + 1; j < n; j++)
        {
if (processes[i].priority < processes[j].priority || (processes[i].priority == processes[j].
            {
                Process temp = processes[i];
                processes[i] = processes[j];
                processes[j] = temp;
            }
        }
    }
    for (int i = 0; i < n; i++)
    {
        if (time < processes[i].arrival_time)
```

```
        {
            time = processes[i].arrival_time;
        }
        printf("%d|%d|", time, processes[i].id);
        processes[i].waiting_time = time - processes[i].arrival_time;
        time += processes[i].burst_time;
        processes[i].turnaround_time = processes[i].waiting_time + processes[i].burst_time;
         printf("%d ---> ", time);
    }
}

void round_robin(Process processes[], int n, int quantum)
{
    int time = processes[0].arrival_time;
    int remaining_burst_times[n];
    for (int i = 0; i < n; i++)
    {
        remaining_burst_times[i] = processes[i].burst_time;
    }
    while (1)
    {
        int done = 1;
        for (int i = 0; i < n; i++)
        {
            if (remaining_burst_times[i] > 0)
            {
                done = 0;
                if (remaining_burst_times[i] > quantum)
                {
                    printf("%d|%d|", time, processes[i].id);
                    time += quantum;
                    remaining_burst_times[i] -= quantum;
                    printf("%d --->", time);
                }
                else
                {
                    printf("%d|%d|", time, processes[i].id);
                    time += remaining_burst_times[i];
                    processes[i].waiting_time = time - processes[i].arrival_time - processes
                    processes[i].turnaround_time = processes[i].waiting_time + processes[i]
                    remaining_burst_times[i] = 0;
                    printf("%d ---> ", time);
                }
            }
        }
        if (done)
```

```
        {
            break;
        }
    }
}

int main()
{
    int n;
    double fcfs,sjf,srtf,prio,rr;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    Process processes[n];
    printf("Enter arrival time, burst time and priority for processs\n");
    for (int i = 0; i < n; i++)
    {
        printf("p%i : ", i+1);
        processes[i].id = i + 1;
        scanf("%d %d %d", &processes[i].arrival_time, &processes[i].burst_time, &processes[i
    }
    printf("\n\nFCFS\n\n");
    FCFS(processes, n);
    printf("Waiting time: %.2f\n", fcfs=calculate_avg_waiting_time(processes, n));
    printf("\n\nSJF\n\n");
    SJF(processes, n);
    printf("Waiting time: %.2f\n", sjf=calculate_avg_waiting_time(processes, n));
    printf("\n\nSRTF\n\n");
    SRTF(processes, n);
    printf("Waiting time: %.2f\n", srtf=calculate_avg_waiting_time(processes, n));
    printf("\n\nPRIORITY SCHEDULING\n\n");
    priority_scheduling(processes, n);
    printf("Waiting time: %.2f\n", prio=calculate_avg_waiting_time(processes, n));
    printf("\n\nROUND ROBIN\n\n");
    round_robin(processes, n, 3);
    printf("Waiting time: %.2f\n", rr=calculate_avg_waiting_time(processes, n));
    if(fcfs <= sjf && fcfs <= srtf && fcfs <= prio && fcfs <= rr){
        printf("\nFCFS has the lowest waiting time at %f s\n",fcfs);
    }
    else if(fcfs >= sjf && sjf <= srtf && sjf <= prio && sjf <= rr){
        printf("\nSJF has the lowest waiting time at %f s\n",sjf);
    }
    else if(srtf <= sjf && fcfs >= srtf && srtf <= prio && srtf <= rr){
        printf("\nSRTF has the lowest waiting time at %f s\n",srtf);
    }
    else if(prio <= sjf && prio <= srtf && fcfs >= prio && prio <= rr){
        printf("\nPriority Scheduling has the lowest waiting time at %f s\n",prio);
```

```
    }
    else{
        printf("\nRound Robin with quatum 2 has the lowest waiting time at %f s\n",rr);
    }
    return 0;
}
```

1 Sample run of the program

```
s23a40@Server-2:~/blab$ gcc expe16.c
s23a40@Server-2:~/blab$ ./a.out
Enter number of processes: 3
Enter arrival time, burst time and priority for processs
p1 : 2
4
1
p2 : 3
2
2
p3 : 5
1
3


FCFS

2|1|6 ---> 6|2|8 ---> 8|3|9 ---> Waiting time: 2.00


SJF

5|3|6 ---> 6|2|8 ---> 8|1|12 ---> Waiting time: 3.00


SRTF

2|1|3 ---> 3|2|4 ---> 4|2|5 ---> 5|3|6 ---> 6|1|7 ---> 7|1|8 ---> 8|1|9 ---> Waiting time: 1.00


PRIORITY SCHEDULING

5|3|6 ---> 6|2|8 ---> 8|1|12 ---> Waiting time: 3.00


ROUND ROBIN

5|3|6 ---> 6|2|8 ---> 8|1|11 --->11|1|12 ---> Waiting time: 3.00

SRTF has the lowest waiting time at 1.000000 s
s23a40@Server-2:~/blab$
```