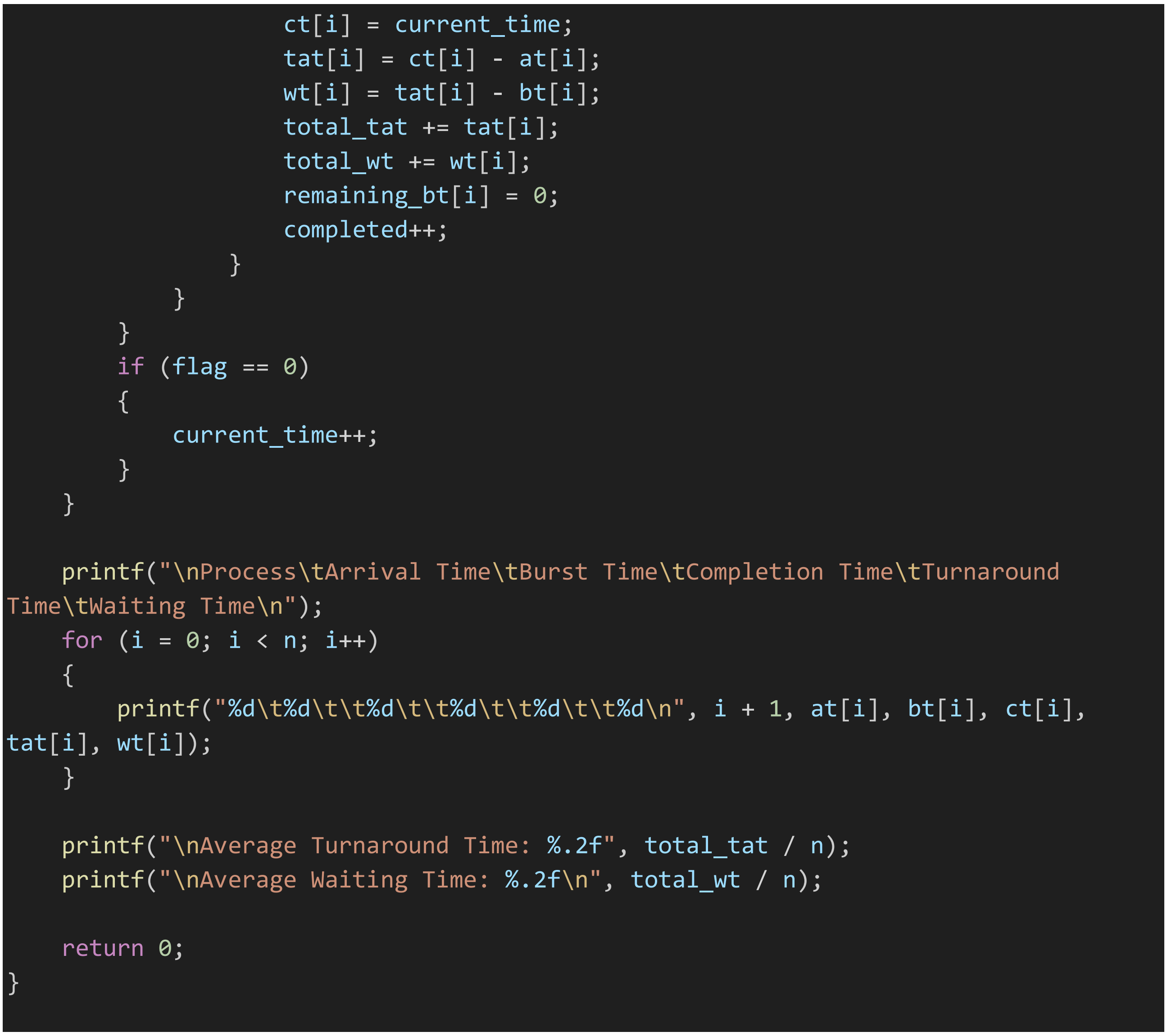
# LAB-02

## Exercise:

1) Implement the Round Robin code and paste the output below.

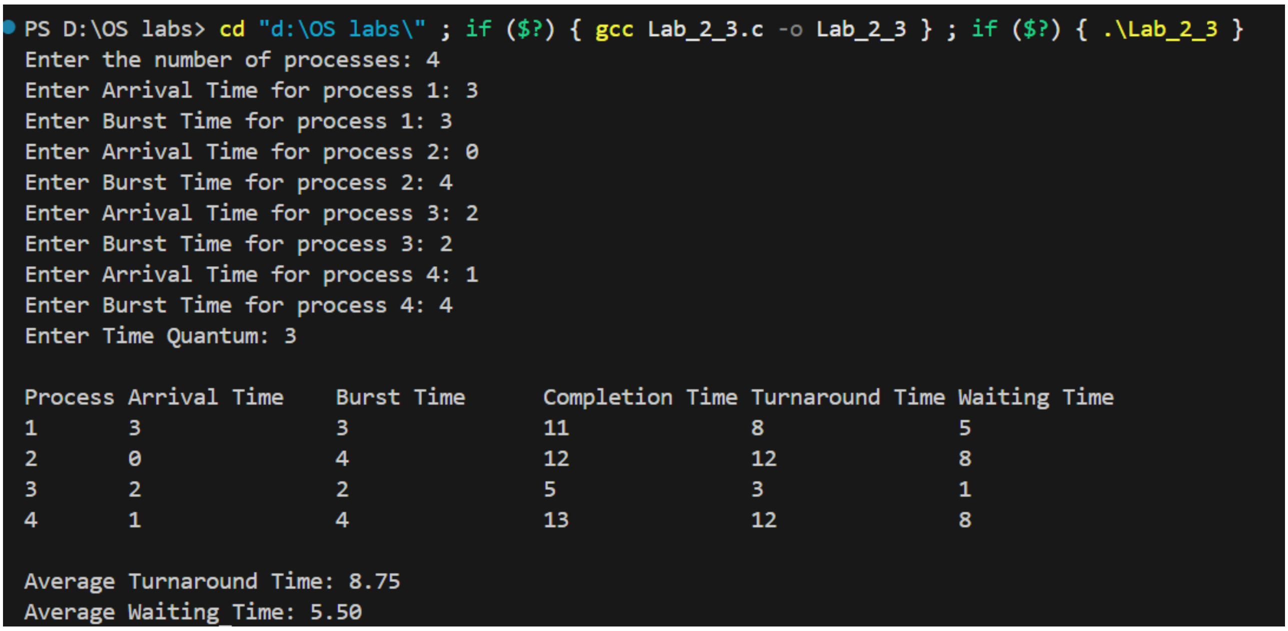
**CODE:**

```
#include <stdio.h>

int main()
{
    int i, n, time_quantum, t = 0;
    int bt[10], at[10], ct[10], tat[10], wt[10], remaining_bt[10];
    float total_tat = 0, total_wt = 0;

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

    for (i = 0; i < n; i++)
    {
        printf("Enter Arrival Time for process %d: ", i + 1);
        scanf("%d", &at[i]);
        printf("Enter Burst Time for process %d: ", i + 1);
        scanf("%d", &bt[i]);
        remaining_bt[i] = bt[i];
    }

    printf("Enter Time Quantum: ");
    scanf("%d", &time_quantum);

    int completed = 0;
    int current_time = 0;
    int flag;

    while (completed < n)
    {
        flag = 0;
        for (i = 0; i < n; i++)
        {
            if (remaining_bt[i] > 0 && at[i] <= current_time)
            {
                flag = 1;
                if (remaining_bt[i] > time_quantum)
                {
                    current_time += time_quantum;
                    remaining_bt[i] -= time_quantum;
                }
                else
                {
                    current_time += remaining_bt[i];
```



```
                    ct[i] = current_time;
                    tat[i] = ct[i] - at[i];
                    wt[i] = tat[i] - bt[i];
                    total_tat += tat[i];
                    total_wt += wt[i];
                    remaining_bt[i] = 0;
                    completed++;
                }
            }
        }
        if (flag == 0)
        {
            current_time++;
        }
    }

    printf("\nProcess\tArrival Time\tBurst Time\tCompletion Time\tTurnaround
Time\tWaiting Time\n");
    for (i = 0; i < n; i++)
    {
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", i + 1, at[i], bt[i], ct[i],
tat[i], wt[i]);
    }

    printf("\nAverage Turnaround Time: %.2f", total_tat / n);
    printf("\nAverage Waiting Time: %.2f\n", total_wt / n);

    return 0;
}
```

**OUTPUT:**



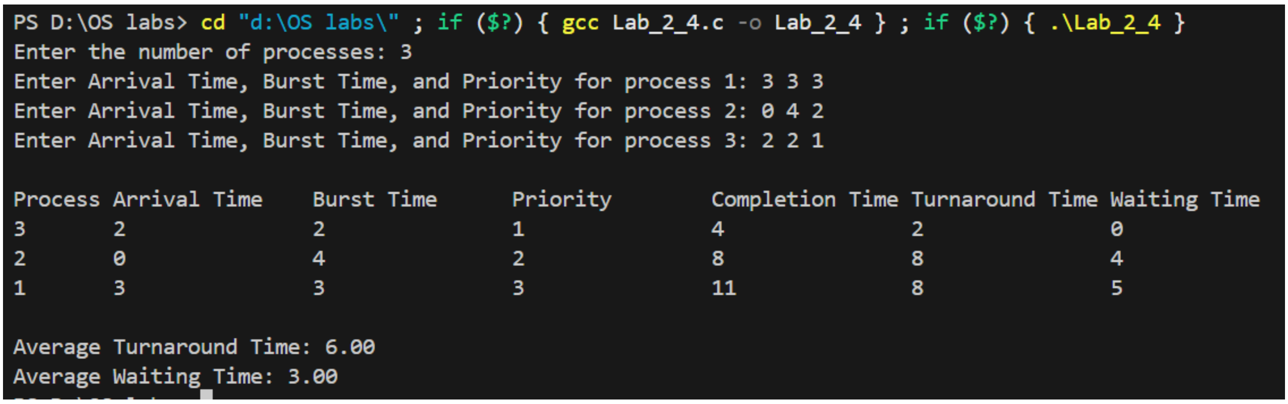
```
PS D:\OS labs> cd "d:\OS labs\" ; if ($?) { gcc Lab_2_3.c -o Lab_2_3 } ; if ($?) { .\Lab_2_3 }
Enter the number of processes: 4
Enter Arrival Time for process 1: 3
Enter Burst Time for process 1: 3
Enter Arrival Time for process 2: 0
Enter Burst Time for process 2: 4
Enter Arrival Time for process 3: 2
Enter Burst Time for process 3: 2
Enter Arrival Time for process 4: 1
Enter Burst Time for process 4: 4
Enter Time Quantum: 3

Process Arrival Time    Burst Time      Completion Time Turnaround Time Waiting Time
1       3               3               11              8               5
2       0               4               12              12              8
3       2               2               5               3               1
4       1               4               13              12              8

Average Turnaround Time: 8.75
Average Waiting Time: 5.50
```

2) Implement the Priority based scheduling code and paste the output below.

**CODE:**

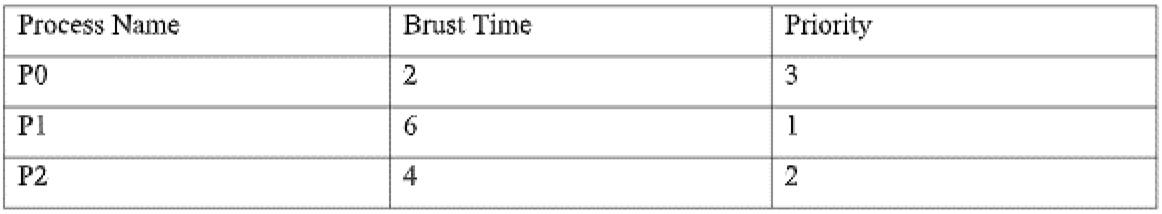
```
#include <stdio.h>

int main()
{
    int n, i, j, temp;
    int p[20], at[20], bt[20], pri[20], ct[20], wt[20], tat[20];
    float total_wt = 0, total_tat = 0;

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

    for (i = 0; i < n; i++)
    {
        p[i] = i + 1;
        printf("Enter Arrival Time, Burst Time, and Priority for process %d: ", i +
1);
        scanf("%d %d %d", &at[i], &bt[i], &pri[i]);
    }

    for (i = 0; i < n; i++)
    {
        for (j = i + 1; j < n; j++)
        {
            if (pri[i] > pri[j])
            {
                temp = pri[i];
                pri[i] = pri[j];
                pri[j] = temp;

                temp = bt[i];
                bt[i] = bt[j];
                bt[j] = temp;

                temp = at[i];
                at[i] = at[j];
                at[j] = temp;

                temp = p[i];
                p[i] = p[j];
                p[j] = temp;
            }
        }
    }

    int current_time = 0;
    for (i = 0; i < n; i++)
    {
        if (current_time < at[i])
        {
            current_time = at[i];
        }
        ct[i] = current_time + bt[i];
        current_time = ct[i];

        tat[i] = ct[i] - at[i];
        wt[i] = tat[i] - bt[i];

        total_tat += tat[i];
        total_wt += wt[i];
    }

    printf("\nProcess\tArrival Time\tBurst Time\tPriority\tCompletion 
Time\tTurnaround Time\tWaiting Time\n");
    for (i = 0; i < n; i++)
    {
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", p[i], at[i], bt[i], 
pri[i], ct[i], tat[i], wt[i]);
    }

    printf("\nAverage Turnaround Time: %.2f", total_tat / n);
    printf("\nAverage Waiting Time: %.2f\n", total_wt / n);

    return 0;
}
```

## OUTPUT:



```
PS D:\OS labs> cd "d:\OS labs\" ; if ($?) { gcc Lab_2_4.c -o Lab_2_4 } ; if ($?) { .\Lab_2_4 }
Enter the number of processes: 3
Enter Arrival Time, Burst Time, and Priority for process 1: 3 3 3
Enter Arrival Time, Burst Time, and Priority for process 2: 0 4 2
Enter Arrival Time, Burst Time, and Priority for process 3: 2 2 1

Process Arrival Time    Burst Time      Priority        Completion Time Turnaround Time Waiting Time
3       2               2               1               4               2               0
2       0               4               2               8               8               4
1       3               3               3               11              8               5

Average Turnaround Time: 6.00
Average Waiting Time: 3.00
```

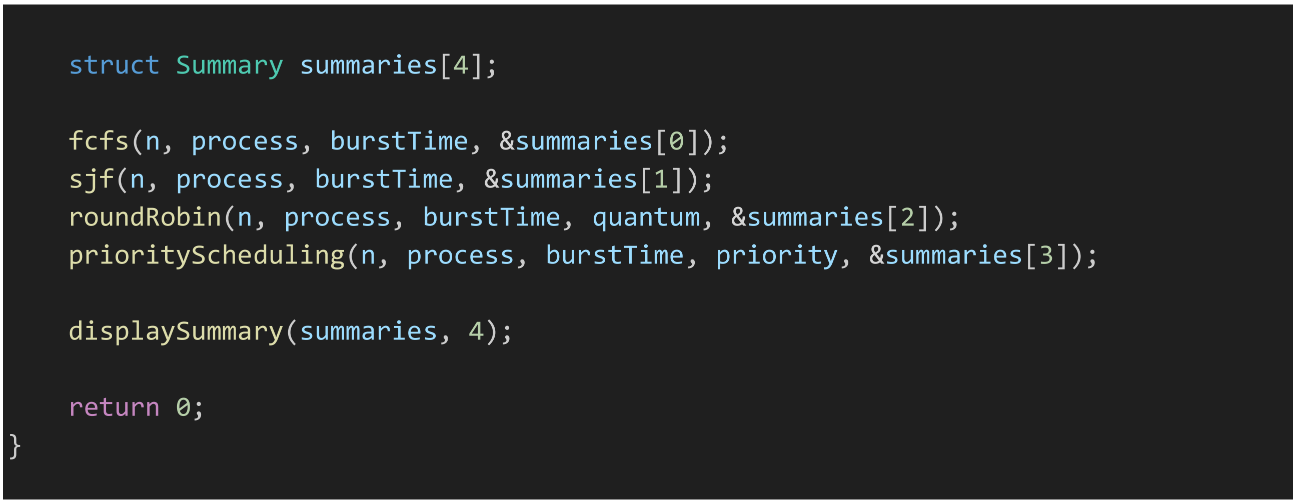
5) Execute all scheduling algorithms on following data and find out the Average Waiting Time and Average Turnaround Time of all scheduling algorithms and discuss your results.
(Quantum Value is 3)



| Process Name | Brust Time | Priority |
|---|---|---|
| P0 | 2 | 3 |
| P1 | 6 | 1 |
| P2 | 4 | 2 |

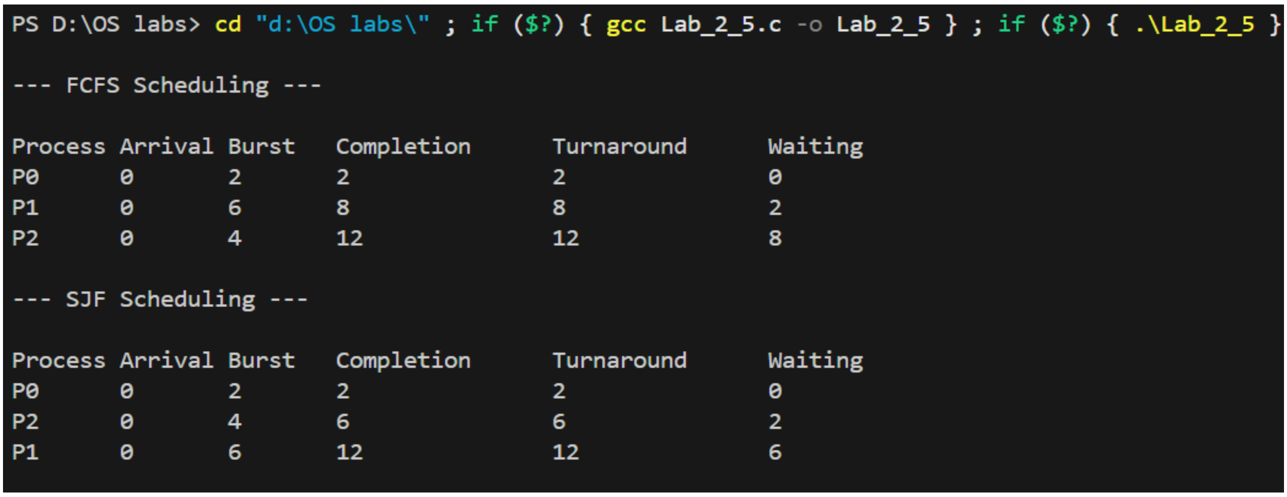
**CODE:**

```
#include <stdio.h>
#include <string.h>
struct Summary
{
    char algorithm[20];
    float avgTurnaroundTime;
    float avgWaitingTime;
};
// Function to display the table
void displayTable(int n, int process[], int burstTime[], int waitingTime[], int
turnaroundTime[], int completionTime[])
{
    printf("\nProcess\tArrival\tBurst\tCompletion\tTurnaround\tWaiting\n");
    for (int i = 0; i < n; i++)
    {
        printf("P%d\t0\t%d\t%d\t\t%d\t\t%d\n", process[i], burstTime[i],
completionTime[i], turnaroundTime[i], waitingTime[i]);
    }
}
// Function to display the summary table
void displaySummary(struct Summary summaries[], int count)
{
    printf("\n--- Scheduling Algorithms Summary ---\n");
    printf("Algorithm\t\tAverage Turnaround Time\tAverage Waiting Time\n");
    for (int i = 0; i < count; i++)
    {
        printf("%-15s\t\t%.2f\t\t\t%.2f\n", summaries[i].algorithm,
summaries[i].avgTurnaroundTime, summaries[i].avgWaitingTime);
    }
}
// FCFS Scheduling
void fcfs(int n, int process[], int burstTime[], struct Summary *summary)
{
    int waitingTime[n], turnaroundTime[n], completionTime[n];
    int totalWT = 0, totalTAT = 0;

    completionTime[0] = burstTime[0];
    turnaroundTime[0] = completionTime[0];
    waitingTime[0] = 0;
    for (int i = 1; i < n; i++)
    {
        completionTime[i] = completionTime[i - 1] + burstTime[i];
        turnaroundTime[i] = completionTime[i];
        waitingTime[i] = turnaroundTime[i] - burstTime[i];
    }

    for (int i = 0; i < n; i++)
    {
        totalWT += waitingTime[i];
        totalTAT += turnaroundTime[i];
    }

    printf("\n--- FCFS Scheduling ---\n");
    displayTable(n, process, burstTime, waitingTime, turnaroundTime,
completionTime);

    summary->avgTurnaroundTime = (float)totalTAT / n;
    summary->avgWaitingTime = (float)totalWT / n;
    strcpy(summary->algorithm, "FCFS");
}
// SJF Scheduling
void sjf(int n, int process[], int burstTime[], struct Summary *summary)
{
    int waitingTime[n], turnaroundTime[n], completionTime[n];
    int sortedProcesses[n], sortedBurst[n];
    int totalWT = 0, totalTAT = 0;

    for (int i = 0; i < n; i++)
    {
        sortedProcesses[i] = process[i];
        sortedBurst[i] = burstTime[i];
    }

    // Sort based on burst time
    for (int i = 0; i < n - 1; i++)
    {
        for (int j = 0; j < n - i - 1; j++)
        {
            if (sortedBurst[j] > sortedBurst[j + 1])
            {
                int temp = sortedBurst[j];
                sortedBurst[j] = sortedBurst[j + 1];
                sortedBurst[j + 1] = temp;

                temp = sortedProcesses[j];
                sortedProcesses[j] = sortedProcesses[j + 1];
                sortedProcesses[j + 1] = temp;
            }
        }
    }

    completionTime[0] = sortedBurst[0];
    turnaroundTime[0] = completionTime[0];
    waitingTime[0] = 0;

    for (int i = 1; i < n; i++)
    {
        completionTime[i] = completionTime[i - 1] + sortedBurst[i];
        turnaroundTime[i] = completionTime[i];
        waitingTime[i] = turnaroundTime[i] - sortedBurst[i];
    }

    for (int i = 0; i < n; i++)
    {
        totalWT += waitingTime[i];
        totalTAT += turnaroundTime[i];
    }

    printf("\n--- SJF Scheduling ---\n");
    displayTable(n, sortedProcesses, sortedBurst, waitingTime, turnaroundTime,
completionTime);

    summary->avgTurnaroundTime = (float)totalTAT / n;
    summary->avgWaitingTime = (float)totalWT / n;
    strcpy(summary->algorithm, "SJF");
}
// Round Robin Scheduling
void roundRobin(int n, int process[], int burstTime[], int quantum, struct Summary
*summary)
{
    int remainingBurst[n], waitingTime[n], turnaroundTime[n], completionTime[n];
    int totalWT = 0, totalTAT = 0;
    int time = 0, done;

    for (int i = 0; i < n; i++)
    {
        remainingBurst[i] = burstTime[i];
        completionTime[i] = 0;
    }

    do
    {
        done = 1;
        for (int i = 0; i < n; i++)
        {
            if (remainingBurst[i] > 0)
            {
                done = 0;
                if (remainingBurst[i] > quantum)
                {
                    time += quantum;
                    remainingBurst[i] -= quantum;
                }
                else
                {
                    time += remainingBurst[i];
                    completionTime[i] = time;
                    remainingBurst[i] = 0;
                }
            }
        }
    } while (!done);

    for (int i = 0; i < n; i++)
    {
        turnaroundTime[i] = completionTime[i];
        waitingTime[i] = turnaroundTime[i] - burstTime[i];
        totalWT += waitingTime[i];
        totalTAT += turnaroundTime[i];
    }

    printf("\n--- Round Robin Scheduling ---\n");
    displayTable(n, process, burstTime, waitingTime, turnaroundTime,
completionTime);

    summary->avgTurnaroundTime = (float)totalTAT / n;
    summary->avgWaitingTime = (float)totalWT / n;
    strcpy(summary->algorithm, "Round Robin");
}
// Priority Scheduling
void priorityScheduling(int n, int process[], int burstTime[], int priority[],
struct Summary *summary)
{
    int waitingTime[n], turnaroundTime[n], completionTime[n];
    int sortedProcesses[n], sortedBurst[n], sortedPriority[n];
    int totalWT = 0, totalTAT = 0;

    for (int i = 0; i < n; i++)
    {
        sortedProcesses[i] = process[i];
        sortedBurst[i] = burstTime[i];
        sortedPriority[i] = priority[i];
    }

    // Sort based on priority
    for (int i = 0; i < n - 1; i++)
    {
        for (int j = 0; j < n - i - 1; j++)
        {
            if (sortedPriority[j] > sortedPriority[j + 1])
            {
                int temp = sortedPriority[j];
                sortedPriority[j] = sortedPriority[j + 1];
                sortedPriority[j + 1] = temp;

                temp = sortedBurst[j];
                sortedBurst[j] = sortedBurst[j + 1];
                sortedBurst[j + 1] = temp;

                temp = sortedProcesses[j];
                sortedProcesses[j] = sortedProcesses[j + 1];
                sortedProcesses[j + 1] = temp;
            }
        }
    }

    completionTime[0] = sortedBurst[0];
    turnaroundTime[0] = completionTime[0];
    waitingTime[0] = 0;

    for (int i = 1; i < n; i++)
    {
        completionTime[i] = completionTime[i - 1] + sortedBurst[i];
        turnaroundTime[i] = completionTime[i];
        waitingTime[i] = turnaroundTime[i] - sortedBurst[i];
    }

    for (int i = 0; i < n; i++)
    {
        totalWT += waitingTime[i];
        totalTAT += turnaroundTime[i];
    }

    printf("\n--- Priority Scheduling ---\n");
    displayTable(n, sortedProcesses, sortedBurst, waitingTime, turnaroundTime,
completionTime);

    summary->avgTurnaroundTime = (float)totalTAT / n;
    summary->avgWaitingTime = (float)totalWT / n;
    strcpy(summary->algorithm, "Priority");
}

int main()
{
    int n = 3;
    int process[] = {0, 1, 2};
    int burstTime[] = {2, 6, 4};
    int priority[] = {3, 1, 2};
    int quantum = 3;
```

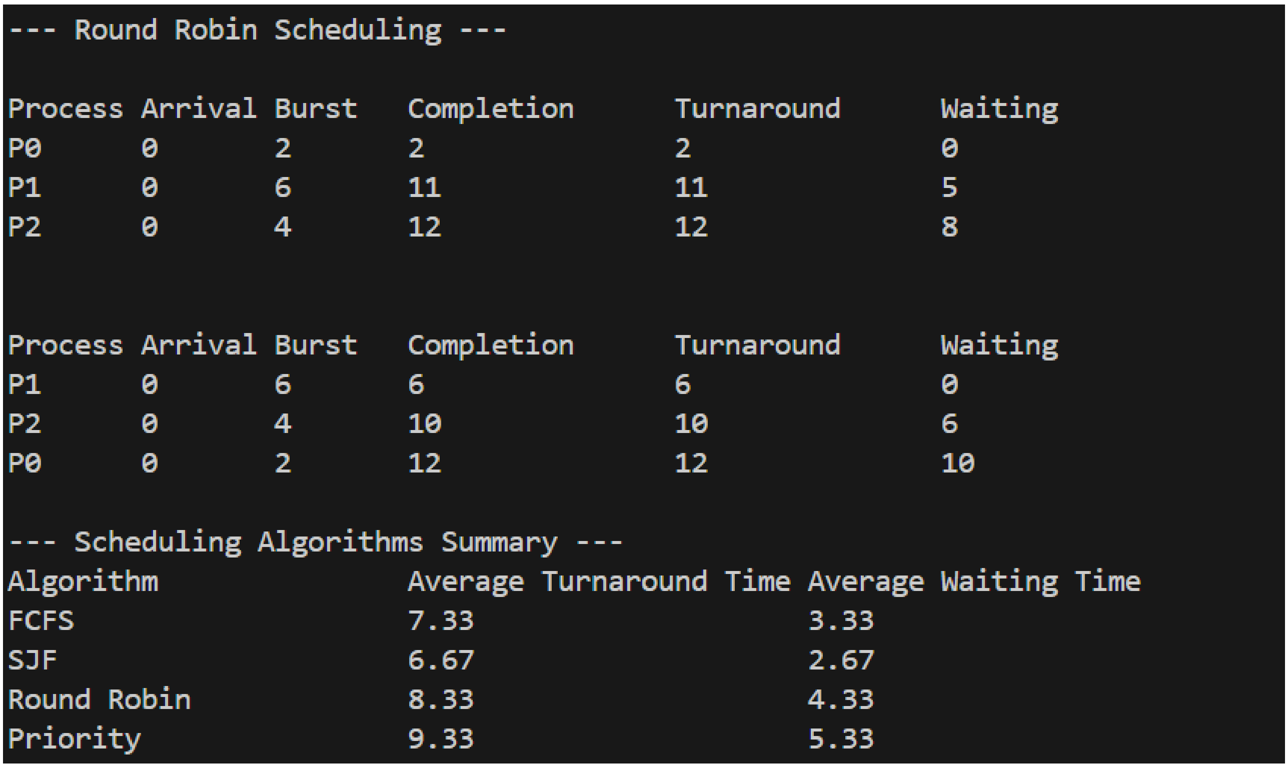


```
    struct Summary summaries[4];

    fcfs(n, process, burstTime, &summaries[0]);
    sjf(n, process, burstTime, &summaries[1]);
    roundRobin(n, process, burstTime, quantum, &summaries[2]);
    priorityScheduling(n, process, burstTime, priority, &summaries[3]);

    displaySummary(summaries, 4);

    return 0;
}
```

**OUTPUT:**



```
PS D:\OS labs> cd "d:\OS labs\" ; if ($?) { gcc Lab_2_5.c -o Lab_2_5 } ; if ($?) { .\Lab_2_5 }

--- FCFS Scheduling ---

Process Arrival Burst   Completion      Turnaround      Waiting
P0      0       2       2               2               0
P1      0       6       8               8               2
P2      0       4       12              12              8

--- SJF Scheduling ---

Process Arrival Burst   Completion      Turnaround      Waiting
P0      0       2       2               2               0
P2      0       4       6               6               2
P1      0       6       12              12              6
```



```
--- Round Robin Scheduling ---

Process Arrival Burst   Completion      Turnaround      Waiting
P0      0       2       2               2               0
P1      0       6       11              11              5
P2      0       4       12              12              8


Process Arrival Burst   Completion      Turnaround      Waiting
P1      0       6       6               6               0
P2      0       4       10              10              6
P0      0       2       12              12              10

--- Scheduling Algorithms Summary ---
Algorithm               Average Turnaround Time Average Waiting Time
FCFS                    7.33                    3.33
SJF                     6.67                    2.67
Round Robin             8.33                    4.33
Priority                9.33                    5.33
```