

NAME: G. GANESH

REG NO: 192373008

EXERICSE-5

Construct a scheduling program with C that selects the waiting process with the highest priority to execute next.

Aim:

To construct a CPU scheduling program in C that uses priority scheduling, where the process with the highest priority is executed next.

Algorithm:

1. Input the number of processes, their burst times, and their priorities.
2. Initialize all processes as unvisited (not yet executed).
3. Select the process with the highest priority (smallest numerical value) from the set of unvisited processes to execute next.
4. Calculate the waiting time and turnaround time for the selected process.
5. Repeat until all processes are executed.
6. Compute the average waiting time and turnaround time.
7. Display the process execution order, waiting times, turnaround times, and averages.

Procedure:

1. Read the burst times and priorities for all processes.
2. Implement priority scheduling by selecting the process with the highest priority iteratively.
3. Track waiting times and turnaround times during execution.
4. Calculate average waiting and turnaround times.
5. Display the results.

Code:

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

```
int main() {
```

```
    int n, i, j, completed = 0, time = 0, highest_priority_index;
```

```
    printf("Enter the number of processes: ");
```

```
    scanf("%d", &n);
```

```
    int burst_time[n], priority[n], waiting_time[n], turnaround_time[n], visited[n];
```

```

float avg_waiting_time = 0, avg_turnaround_time = 0;

printf("Enter the burst times and priorities of the processes:\n");

for (i = 0; i < n; i++) {
    printf("Process %d - Burst Time: ", i + 1);
    scanf("%d", &burst_time[i]);
    printf("Process %d - Priority: ", i + 1);
    scanf("%d", &priority[i]);
    visited[i] = 0; // Mark all processes as unvisited initially
}

while (completed < n) {
    int highest_priority = 1e9;
    highest_priority_index = -1;
    for (i = 0; i < n; i++) {
        if (!visited[i] && priority[i] < highest_priority) {
            highest_priority = priority[i];
            highest_priority_index = i;
        }
    }

    if (highest_priority_index != -1) {
        waiting_time[highest_priority_index] = time;
        time += burst_time[highest_priority_index];
        turnaround_time[highest_priority_index] = time;
        visited[highest_priority_index] = 1;
        completed++;
    }
}

for (i = 0; i < n; i++) {
    avg_waiting_time += waiting_time[i];
    avg_turnaround_time += turnaround_time[i];
}

```

```

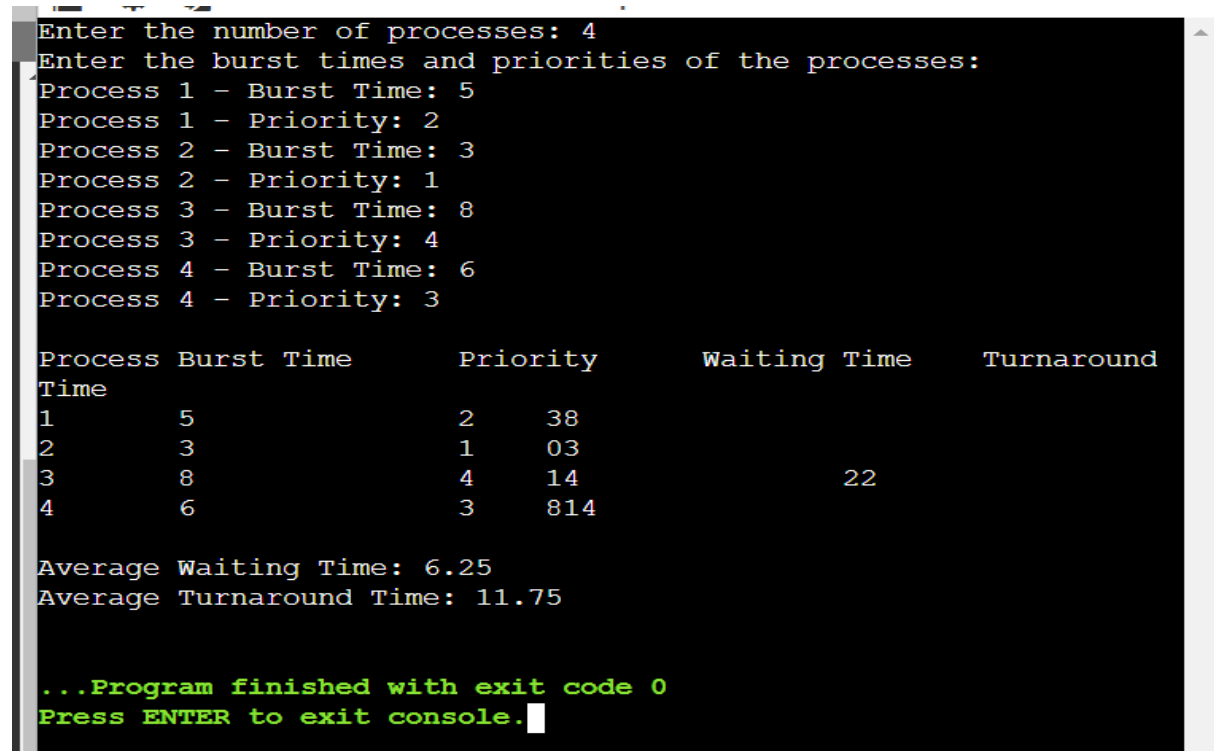
    avg_waiting_time /= n;
    avg_turnaround_time /= n;
    printf("\nProcess\tBurst Time\tPriority\tWaiting Time\tTurnaround Time\n");
    for (i = 0; i < n; i++) {
        printf("%d\t%d\t%d\t%d\t%d\t%d\n", i + 1, burst_time[i], priority[i], waiting_time[i],
turnaround_time[i]);
    }
    printf("\nAverage Waiting Time: %.2f\n", avg_waiting_time);
    printf("Average Turnaround Time: %.2f\n", avg_turnaround_time);
    return 0;
}

```

Result:

The priority scheduling program successfully selects the process with the highest priority (lowest numerical value) for execution, calculates waiting and turnaround times, and computes the averages. The output displays the execution order and timing details for all processes.

Output:



```

Enter the number of processes: 4
Enter the burst times and priorities of the processes:
Process 1 - Burst Time: 5
Process 1 - Priority: 2
Process 2 - Burst Time: 3
Process 2 - Priority: 1
Process 3 - Burst Time: 8
Process 3 - Priority: 4
Process 4 - Burst Time: 6
Process 4 - Priority: 3

Process Burst Time      Priority      Waiting Time      Turnaround
Time
1          5           2           38
2          3           1           03
3          8           4           14
4          6           3           814

Average Waiting Time: 6.25
Average Turnaround Time: 11.75

...Program finished with exit code 0
Press ENTER to exit console.

```

