Experiment:6-Construct a C program to implement preemptive priority scheduling algorithm

Aim:

The aim of this program is to implement Preemptive Priority Scheduling in C. In preemptive priority scheduling, the CPU is assigned to the process with the highest priority (smallest priority number). If a new process arrives with a higher priority (lower priority number) than the currently running process, the current process is preempted and the new process starts executing.

Procedure:

1. Input: Get the number of processes and their details (burst time, priority).
2. Sort processes based on priority and burst time.
3. Execution: The process with the highest priority is executed first.
4. Preemption: If a new process with a higher priority arrives while another process is running, the running process will be preempted.
5. Calculate the waiting time and turnaround time for each process.
6. Output the process details including their waiting time, turnaround time, and the averages for all processes.

Preemptive Priority Scheduling Algorithm:

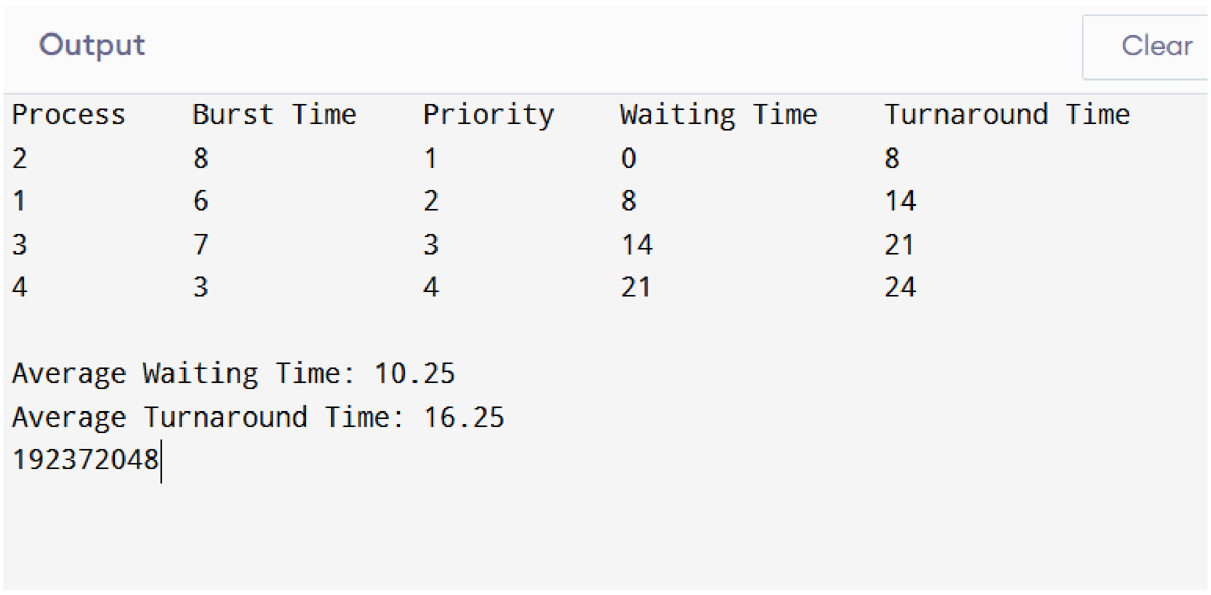
- Preemption means that if a process with a higher priority arrives during the execution of the current process, the current process is paused, and the new process starts executing.
- The priority determines the order in which processes are executed, where a smaller number indicates a higher priority.

C Program Implementation:

```
#include <stdio.h>


struct Process {

   int id;

   int burst_time;

   int priority;

   int remaining_time;

   int waiting_time;

   int turnaround_time;

};

void sortByPriority(struct Process processes[], int n) {

   struct Process temp;
```

```
    for (int i = 0; i < n-1; i++) {
        for (int j = i+1; j < n; j++) {
            if (processes[i].priority > processes[j].priority) {
                temp = processes[i];
                processes[i] = processes[j];
                processes[j] = temp;
            }
        }
    }
}
int main() {
    int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    struct Process processes[n];
    int total_waiting_time = 0, total_turnaround_time = 0;
    for (int i = 0; i < n; i++) {
        processes[i].id = i + 1;
        printf("Enter burst time and priority for process %d: ", i + 1);
        scanf("%d %d", &processes[i].burst_time, &processes[i].priority);
        processes[i].remaining_time = processes[i].burst_time; // Initialize remaining time
    }
    sortByPriority(processes, n);
    int current_time = 0;
    int completed = 0;
    while (completed < n) {
        int highest_priority_idx = -1;
        int min_priority = 9999;


        for (int i = 0; i < n; i++) {
            if (processes[i].remaining_time > 0 && processes[i].priority < min_priority) {
                min_priority = processes[i].priority;
                highest_priority_idx = i;
            }
        }
        if (highest_priority_idx != -1) {
            processes[highest_priority_idx].remaining_time--;
            current_time++;
            if (processes[highest_priority_idx].remaining_time == 0) {
                processes[highest_priority_idx].turnaround_time = current_time;
                processes[highest_priority_idx].waiting_time = 
processes[highest_priority_idx].turnaround_time - processes[highest_priority_idx].burst_time;
                total_waiting_time += processes[highest_priority_idx].waiting_time;
                total_turnaround_time += processes[highest_priority_idx].turnaround_time;
                completed++;
            }
        }
    }
    printf("\nProcess\tBurst Time\tPriority\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\n", processes[i].id, processes[i].burst_time, 
processes[i].priority,
               processes[i].waiting_time, processes[i].turnaround_time);
    }
    printf("\nAverage Waiting Time: %.2f\n", (float)total_waiting_time / n);
    printf("Average Turnaround Time: %.2f\n", (float)total_turnaround_time / n);
    return 0;
}
```

Output:



```
Process     Burst Time    Priority    Waiting Time    Turnaround Time
2           8             1           0               8
1           6             2           8               14
3           7             3           14              21
4           3             4           21              24

Average Waiting Time: 10.25
Average Turnaround Time: 16.25
192372048
```

Result:

This program implements **Preemptive Priority Scheduling** where the process with the highest priority is always selected for execution. If a new process with a higher priority arrives, the current process is preempted and the higher-priority process is executed. The program calculates the waiting time, turnaround time, and averages for all processes.