Experiment: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 selects the waiting process with the highest priority to execute next. This scheduling algorithm is Preemptive Priority Scheduling, where processes are assigned priorities, and the process with the highest priority (the smallest priority number) is selected for execution first.

Procedure:

1. Input the number of processes, their burst times, and their priorities.
2. Sort the processes by their priorities (highest priority first).
3. For each process, calculate its waiting time (time from arrival until it starts executing).
4. Calculate the turnaround time (waiting time + burst time) for each process.
5. Output the waiting time and turnaround time for each process.
6. Calculate the average waiting time and average turnaround time.

Preemptive Priority Scheduling:

- The highest priority process (smallest priority number) is selected to run next.
- If a new process arrives with a higher priority during the execution of a process, the running process is preempted, and the new process starts execution.

C Program Implementation:

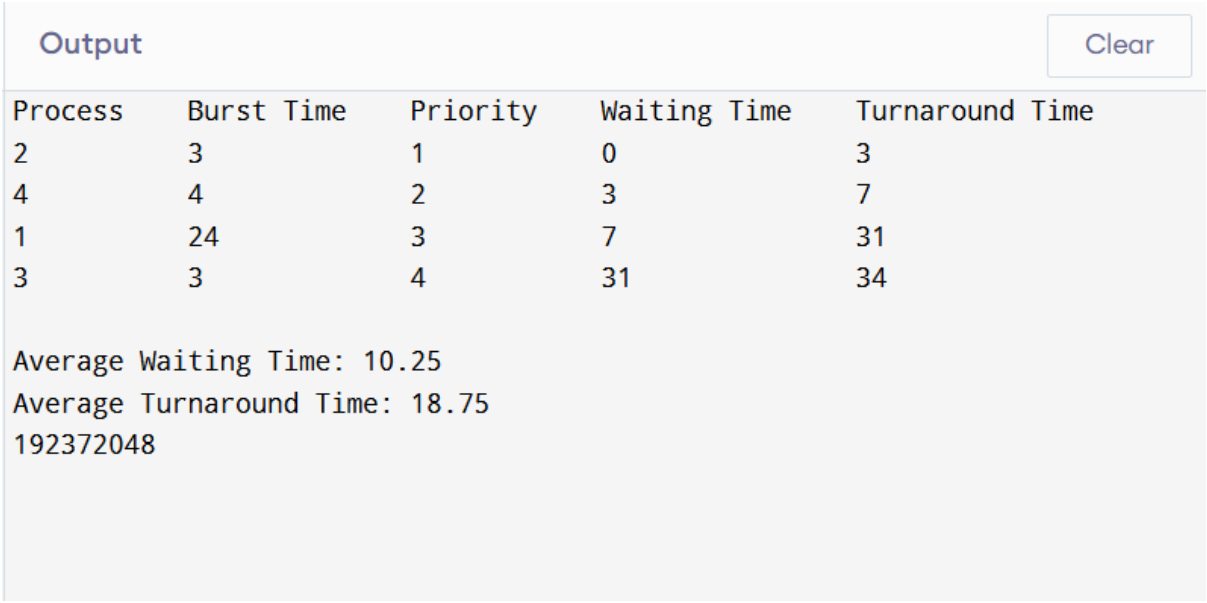
```
#include <stdio.h>


struct Process {
   int id;
   int burst_time;
   int waiting_time;
   int turnaround_time;
   int priority;
   int remaining_time;  // Used to track remaining burst time for preemption
};


void sortByPriority(struct Process processes[], int n) {
   // Sort the processes based on priority (lowest priority number is highest priority)
```

```
    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;

    // Input burst time and priority for each process
    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
    }

    // Sort processes based on priority
    sortByPriority(processes, n);
```

```
    int current_time = 0;
    int completed = 0;
    int remaining_processes = n;

    // Execute processes according to priority
    while (completed < n) {
        for (int i = 0; i < n; i++) {
            if (processes[i].remaining_time > 0) {
                // If process is not completed yet, execute it
                processes[i].remaining_time--;
                current_time++;

                // If process finishes
                if (processes[i].remaining_time == 0) {
                    processes[i].turnaround_time = current_time;
                    processes[i].waiting_time = processes[i].turnaround_time - processes[i].burst_time;
                    total_waiting_time += processes[i].waiting_time;
                    total_turnaround_time += processes[i].turnaround_time;
                    completed++;
                }
            }
        }
    }

    // Output results
    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:



Result:

This program implements **Priority Scheduling** where the CPU selects the process with the highest priority (lowest priority number) for execution. The program calculates the waiting time and turnaround time for each process and computes the averages.