192311035

M.RAVI SAI VINAY

EXPERIMENT-6

Construct a C program to implement preemptive priority scheduling algorithm

AIM:-

To design a C program that implements the Preemptive Priority Scheduling Algorithm, where the CPU is allocated to the process with the highest priority (lowest priority number), and the CPU may be preempted when a process with a higher priority arrives.

ALGORITHM:-

Input Process Details:

Read the number of processes.

Input the burst times, arrival times, and priorities for each process.

Track Remaining Burst Times:

Maintain a record of remaining burst times for all processes.

Select the Process with Highest Priority:

At each time unit, select the process with the highest priority (lowest priority number) that has arrived and has remaining burst time.

Preempt or Continue Execution:

If a new process with a higher priority arrives, preempt the current process.

Calculate Completion Time (CT):

When a process finishes, record its completion time.

Calculate Turnaround Time (TAT):

TAT = CT - Arrival Time.

Calculate Waiting Time (WT):

WT = TAT - Burst Time.

Calculate Average TAT and WT:

Compute the average turnaround time and waiting time.

Display Results:

Show process details, including burst time, arrival time, priority, completion time, turnaround time, and waiting time.

CODE:-

```
#include <stdio.h>

#include <limits.h>


typedef struct {

    int process_id;

    int arrival_time;

    int burst_time;

    int priority;

    int completion_time;

    int turnaround_time;

    int waiting_time;

    int remaining_time;

} Process;


int main() {

    int n, current_time = 0, completed = 0, min_priority_index;

    float avg_turnaround_time = 0, avg_waiting_time = 0;
```

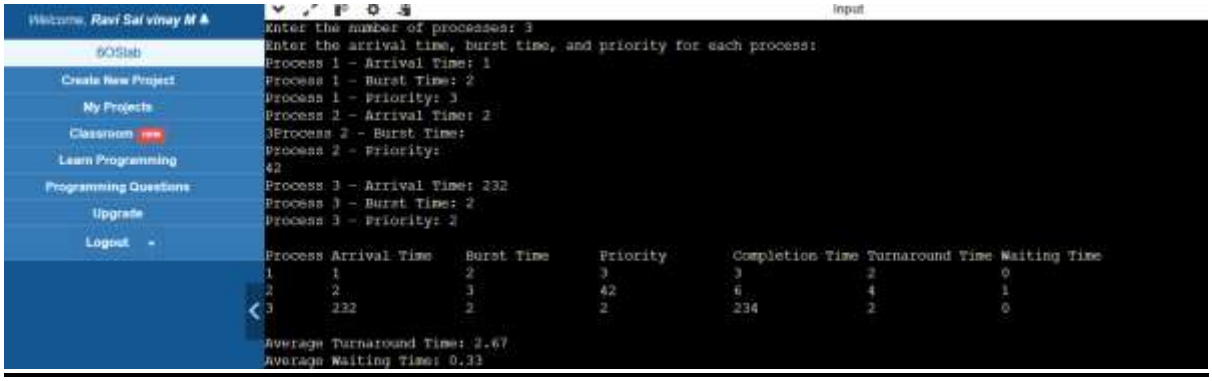
```
printf("Enter the number of processes: ");
scanf("%d", &n);

Process processes[n];

// Input process details
printf("Enter the arrival time, burst time, and priority for each process:\n");
for (int i = 0; i < n; i++) {
    processes[i].process_id = i + 1;
    printf("Process %d - Arrival Time: ", i + 1);
    scanf("%d", &processes[i].arrival_time);
    printf("Process %d - Burst Time: ", i + 1);
    scanf("%d", &processes[i].burst_time);
    printf("Process %d - Priority: ", i + 1);
    scanf("%d", &processes[i].priority);
    processes[i].remaining_time = processes[i].burst_time;
}

// Preemptive Priority Scheduling
while (completed != n) {
    min_priority_index = -1;
    int min_priority = INT_MAX;

    // Find process with the highest priority (lowest number) that has arrived
```

```
        for (int i = 0; i < n; i++) {
            if (processes[i].arrival_time <= current_time && processes[i].remaining_time > 0) {
                if (processes[i].priority < min_priority ||
                    (processes[i].priority == min_priority && processes[i].arrival_time < processes[min_priority_index].arrival_time)) {
                    min_priority = processes[i].priority;
                    min_priority_index = i;
                }
            }
        }

        if (min_priority_index != -1) {
            // Execute the process for one time unit
            processes[min_priority_index].remaining_time--;

            // If the process is completed
            if (processes[min_priority_index].remaining_time == 0) {
                completed++;
                processes[min_priority_index].completion_time = current_time + 1;
                processes[min_priority_index].turnaround_time = processes[min_priority_index].completion_time - processes[min_priority_index].arrival_time;
                processes[min_priority_index].waiting_time = processes[min_priority_index].turnaround_time - processes[min_priority_index].burst_time;
```

```
            avg_turnaround_time += processes[min_priority_index].turnaround_time;
            avg_waiting_time += processes[min_priority_index].waiting_time;
        }
    }

    // Increment time
    current_time++;
}

// Calculate averages
avg_turnaround_time /= n;
avg_waiting_time /= n;

// Display results
printf("\nProcess\tArrival Time\tBurst Time\tPriority\tCompletion Time\tTurnaround Time\tWaiting Time\n");
for (int 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", processes[i].process_id, processes[i].arrival_time,
           processes[i].burst_time, processes[i].priority, processes[i].completion_time,
           processes[i].turnaround_time, processes[i].waiting_time);
}

printf("\nAverage Turnaround Time: %.2f\n", avg_turnaround_time);
printf("Average Waiting Time: %.2f\n", avg_waiting_time);
```

```
    return 0;

}
```

OUTPUT:-



RESULT:-

The Preemptive Priority Scheduling program was successfully implemented in C. The program dynamically preempted the CPU when a process with a higher priority arrived, and it correctly calculated the completion, turnaround, and waiting times for all processes, along with their averages.