B.BHANUTEJA REDDY-192325016

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 process with the highest priority (lowest priority value) is executed first, and the scheduler can preempt a running process if a higher-priority process arrives.

Algorithm:

1. Start the program.
2. Input the number of processes, their burst times, arrival times, and priorities.
3. Initialize variables for tracking time, remaining burst times, and the completed status of processes.
4. At each time unit:
    - Check all arrived processes and select the one with the highest priority (lowest priority value) that has remaining burst time.
    - Execute the selected process for one unit of time.
    - Update its remaining burst time.
5. If a process completes execution, record its completion time and calculate its turnaround time and waiting time.
6. Repeat until all processes are completed.
7. Calculate average waiting time and turnaround time.
8. Display the results.
9. End the program.

Procedure:

1. Include necessary headers: <stdio.h> and <limits.h> (for constants like INT_MAX).
2. Use arrays to store process attributes such as burst times, arrival times, priorities, waiting times, and turnaround times.
3. Implement a loop to simulate the scheduling timeline and dynamically select the highest-priority process.
4. Track completion and compute metrics.

CODE:

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

int main() {
   int n, i, time = 0, completed = 0, min_priority, current_process = -1;
   float avg_wait = 0, avg_turnaround = 0;

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

   int burst_time[n], remaining_time[n], arrival_time[n], priority[n];
   int waiting_time[n], turnaround_time[n], completion_time[n], is_completed[n];

   printf("Enter arrival times, burst times, and priorities for each process:\n");
   for (i = 0; i < n; i++) {
      printf("Process %d:\n", i + 1);
      printf("Arrival Time: ");
      scanf("%d", &arrival_time[i]);
      printf("Burst Time: ");
      scanf("%d", &burst_time[i]);
      printf("Priority: ");
      scanf("%d", &priority[i]);
      remaining_time[i] = burst_time[i];
      is_completed[i] = 0;
   }

   while (completed < n) {
```

```
    min_priority = INT_MAX;
    current_process = -1;

    for (i = 0; i < n; i++) {
      if (arrival_time[i] <= time && !is_completed[i] && priority[i] < min_priority) {
        min_priority = priority[i];
        current_process = i;
      }
    }

    if (current_process == -1) {
      time++;
      continue;
    }

    remaining_time[current_process]--;
    time++;

    if (remaining_time[current_process] == 0) {
      is_completed[current_process] = 1;
      completed++;
      completion_time[current_process] = time;
      turnaround_time[current_process] = completion_time[current_process] -
arrival_time[current_process];
      waiting_time[current_process] = turnaround_time[current_process] -
burst_time[current_process];
      avg_wait += waiting_time[current_process];
      avg_turnaround += turnaround_time[current_process];
    }
```

```
    }

    avg_wait /= n;
    avg_turnaround /= n;

    printf("\nProcess\tArrival Time\tBurst Time\tPriority\tWaiting Time\tTurnaround 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, arrival_time[i], burst_time[i], priority[i], waiting_time[i], turnaround_time[i]);
    }

    printf("\nAverage Waiting Time: %.2f\n", avg_wait);
    printf("Average Turnaround Time: %.2f\n", avg_turnaround);

    return 0;
}
```

OUTPUT: