A. HARSHA MUKUNDHA - 192324070

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

Aim:

To design a CPU scheduling program using the Priority Scheduling technique, which selects the waiting process with the highest priority to execute next.

Algorithm:

1. Start the program.
2. Input the number of processes, their burst times, and their priorities.
3. Sort the processes based on their priorities in ascending order (lower priority value indicates higher priority).
4. Calculate the waiting time for each process:
    - Waiting time for the first process is 0.
    - For subsequent processes, Waiting Time[i] = Waiting Time[i-1] + Burst Time[i-1].
5. Calculate the turnaround time for each process:
    - Turnaround Time[i] = Waiting Time[i] + Burst Time[i].
6. Display the process details, including their burst time, priority, waiting time, and turnaround time.
7. Compute the average waiting time and turnaround time.
8. End the program.

Procedure:

1. Include necessary headers: <stdio.h>.
2. Define arrays for process IDs, burst times, priorities, waiting times, and turnaround times.
3. Sort the processes by priority using a simple sorting algorithm (e.g., Bubble Sort).
4. Compute waiting times and turnaround times iteratively.
5. Calculate and display average waiting and turnaround times.

CODE:

```
#include <stdio.h>

int main() {
   int n, i, j, temp;
   float avg_wait = 0, avg_turnaround = 0;

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

   int process[n], burst_time[n], priority[n], waiting_time[n], turnaround_time[n];

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

   for (i = 0; i < n - 1; i++) {
      for (j = 0; j < n - i - 1; j++) {
         if (priority[j] > priority[j + 1]) {
            temp = priority[j];
            priority[j] = priority[j + 1];
            priority[j + 1] = temp;
```

```
            temp = burst_time[j];
            burst_time[j] = burst_time[j + 1];
            burst_time[j + 1] = temp;

            temp = process[j];
            process[j] = process[j + 1];
            process[j + 1] = temp;
        }
    }
}

waiting_time[0] = 0;
for (i = 1; i < n; i++) {
    waiting_time[i] = waiting_time[i - 1] + burst_time[i - 1];
}

for (i = 0; i < n; i++) {
    turnaround_time[i] = waiting_time[i] + burst_time[i];
    avg_wait += waiting_time[i];
    avg_turnaround += turnaround_time[i];
}

avg_wait /= n;
avg_turnaround /= n;

printf("\nProcess\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\n", process[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: