Ex. No.: 6c)

PRIORITY SCHEDULING

Aim:

To implement priority scheduling technique

Algorithm:

1. Get the number of processes from the user.

2. Read the process name, burst time and priority of process.

3. Sort based on burst time of all processes in ascending order based priority 4.

Calculate the total waiting time and total turnaround time for each process 5.

Display the process name & burst time for each process.

6. Display the total waiting time, average waiting time, turnaround time

Program Code:

Non-Preemptive Priority Scheduling

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

struct Process { char name[10]; int burst_time; int priority; int waiting_time; int
turnaround_time; };

void sortProcesses(struct Process p[], int n) { struct Process temp; for (int i = 0; i < n - 1;
i++) { for (int j = i + 1; j < n; j++) { if (p[i].priority > p[j].priority) { temp = p[i]; p[i] = p[j]; p[j]
= temp; } } } }

void calculatePriorityScheduling(struct Process p[], int n) { int total_wt = 0, total_tat = 0;

// Sort processes based on priority (ascending order)
sortProcesses(p, n);

// Calculate waiting time and turnaround time
p[0].waiting_time = 0;
for (int i = 1; i < n; i++) {
    p[i].waiting_time = p[i - 1].waiting_time + p[i - 1].burst_time;
}

for (int i = 0; i < n; i++) {
    p[i].turnaround_time = p[i].waiting_time + p[i].burst_time;
    total_wt += p[i].waiting_time;
    total_tat += p[i].turnaround_time;
}

// Display results
printf("\nProcess\tBurst Time\tPriority\tWaiting Time\tTurnaround Time\n");
for (int i = 0; i < n; i++) {
    printf("%s\t\t%d\t\t%d\t\t%d\t\t%d\n", p[i].name, p[i].burst_time, p[i].priority,
p[i].waiting_time, p[i].turnaround_time);
}

printf("\nAverage Waiting Time: %.2f", (float)total_wt / n);
printf("\nAverage Turnaround Time: %.2f", (float)total_tat / n);


}

int main() { int n;

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

struct Process p[n];

// Input process details
for (int i = 0; i < n; i++) {
    printf("\nEnter process name (e.g., P1, P2, etc.): ");
    scanf("%s", p[i].name);
    printf("Enter burst time for %s: ", p[i].name);
    scanf("%d", &p[i].burst_time);
    printf("Enter priority for %s (lower value means higher priority): ", p[i].name);
    scanf("%d", &p[i].priority);
}

// Calculate and display results
calculatePriorityScheduling(p, n);

return 0;
```

```
}
```

INPUT:

```
Enter the number of processes: 3

Enter process name (e.g., P1, P2, etc.): P1
Enter burst time for P1: 6
Enter priority for P1 (lower value means higher priority): 2

Enter process name (e.g., P1, P2, etc.): P2
Enter burst time for P2: 4
Enter priority for P2 (lower value means higher priority): 1

Enter process name (e.g., P1, P2, etc.): P3
Enter burst time for P3: 3
Enter priority for P3 (lower value means higher priority): 3
```

OUTPUT:

```
Process    Burst Time    Priority    Waiting Time    Turnaround Time
P2         4             1           0               4
P1         6             2           4               10
P3         3             3           10               13
```

```
Average Waiting Time: 4.67

Average Turnaround Time: 9.00
```

Preemptive Priority Scheduling

```
#include <stdio.h>

 #include <string.h>

struct Process { char name[10]; int burst_time; int remaining_burst_time; int priority; int
waiting_time; int turnaround_time; int completion_time; int start_time; };

void calculatePreemptivePriorityScheduling(struct Process p[], int n) { int total_wt = 0,
total_tat = 0; int current_time = 0; int completed = 0;

// Sort processes based on priority (ascending order)
struct Process temp;
for (int i = 0; i < n - 1; i++) {
    for (int j = i + 1; j < n; j++) {
        if (p[i].priority > p[j].priority) {
            temp = p[i];
            p[i] = p[j];
            p[j] = temp;
        }
    }
}

// Initialize remaining burst times
for (int i = 0; i < n; i++) {
    p[i].remaining_burst_time = p[i].burst_time;
}

// Execute processes (preemptively)
while (completed < n) {
    int min_priority = -1;
    int process_index = -1;

    // Find the process with highest priority and remaining burst time > 0
    for (int i = 0; i < n; i++) {
        if (p[i].remaining_burst_time > 0) {
            if (min_priority == -1 || p[i].priority < min_priority) {
                min_priority = p[i].priority;
                process_index = i;
            }
        }
    }

    // Execute the selected process for 1 unit of time
    p[process_index].remaining_burst_time--;
    current_time++;

    // If process completed execution
    if (p[process_index].remaining_burst_time == 0) {
        p[process_index].completion_time = current_time;
        p[process_index].turnaround_time = p[process_index].completion_time;
        p[process_index].waiting_time = p[process_index].turnaround_time -
p[process_index].burst_time;
        total_wt += p[process_index].waiting_time;
        total_tat += p[process_index].turnaround_time;
        completed++;
    }
}

// Display results
printf("\nProcess\tBurst Time\tPriority\tWaiting Time\tTurnaround Time\n");
for (int i = 0; i < n; i++) {
    printf("%s\t\t%d\t\t%d\t\t%d\t\t%d\n", p[i].name, p[i].burst_time, p[i].priority,
p[i].waiting_time, p[i].turnaround_time);
}

printf("\nAverage Waiting Time: %.2f", (float)total_wt / n);
printf("\nAverage Turnaround Time: %.2f", (float)total_tat / n);


}

int main() { int n;

// Input the number of processes
printf("Enter the number of processes: ");
```

```
scanf("%d", &n);

struct Process p[n];

// Input process details
for (int i = 0; i < n; i++) {
    printf("\nEnter process name (e.g., P1, P2, etc.): ");
    scanf("%s", p[i].name);
    printf("Enter burst time for %s: ", p[i].name);
    scanf("%d", &p[i].burst_time);
    printf("Enter priority for %s (lower value means higher priority): ", p[i].name);
    scanf("%d", &p[i].priority);
}

// Calculate and display results
calculatePreemptivePriorityScheduling(p, n);

return 0;
 }
```

INPUT:

```
Enter the number of processes: 3


Enter process name (e.g., P1, P2, etc.): P1

Enter burst time for P1: 6

Enter priority for P1 (lower value means higher priority): 2


Enter process name (e.g., P1, P2, etc.): P2

Enter burst time for P2: 4

Enter priority for P2 (lower value means higher priority): 1


Enter process name (e.g., P1, P2, etc.): P3

Enter burst time for P3: 3
Enter priority for P3 (lower value means higher priority): 3
```

OUTPUT:

```
Process    Burst Time    Priority    Waiting Time    Turnaround Time
P2          4            1           0               4
P1          6            2           4               10
P3          3            3           10               13

Average Waiting Time: 4.67
Average Turnaround Time: 9.00
```

RESULT:

Program To implement priority scheduling technique is successful.