Uday Krishna N 231901057

Ex. No.: 6c)
Date:19/03/2025

# 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 )

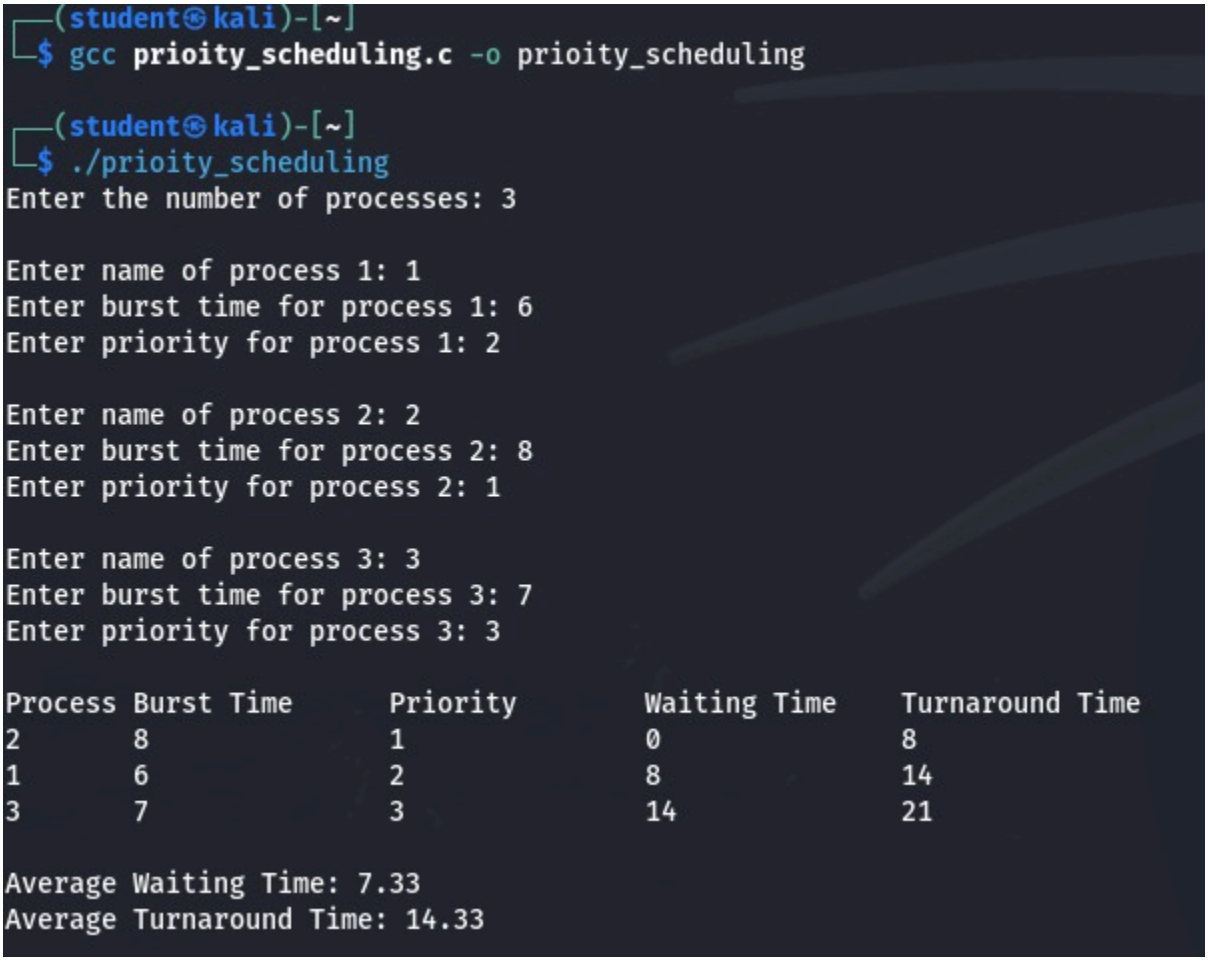
```
#include <stdio.h>

void calculateWaitingTime(int processes[], int n, int burst_time[], int waiting_time[], int priority[]) {
        // Sorting processes based on priority
        int temp;
        for (int i = 0; i < n-1; i++) {
        for (int j = i+1; j < n; j++) {
        if (priority[i] > priority[j]) {
                // Swap priority
                temp = priority[i];
                priority[i] = priority[j];
                priority[j] = temp;
                // Swap burst times
                temp = burst_time[i];
```

```
                burst_time[i] = burst_time[j];
                burst_time[j] = temp;
                // Swap process names
                temp = processes[i];
                processes[i] = processes[j];
                processes[j] = temp;
        }
        }
        }

        // Waiting time of the first process is 0
        waiting_time[0] = 0;

        // Calculate waiting time for all processes
        for (int i = 1; i < n; i++) {
        waiting_time[i] = burst_time[i - 1] + waiting_time[i - 1];
        }
}

void calculateTurnaroundTime(int burst_time[], int waiting_time[], int n, int
turnaround_time[]) {
        // Turnaround time = Burst time + Waiting time
        for (int i = 0; i < n; i++) {
        turnaround_time[i] = burst_time[i] + waiting_time[i];
        }
}

void displayResults(int processes[], int burst_time[], int priority[], int waiting_time[], int
turnaround_time[], int n) {
        int total_waiting_time = 0;
        int total_turnaround_time = 0;

        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], burst_time[i], priority[i],
waiting_time[i], turnaround_time[i]);
```

```
        total_waiting_time += waiting_time[i];
        total_turnaround_time += turnaround_time[i];
        }

        float avg_waiting_time = (float) total_waiting_time / n;
        float avg_turnaround_time = (float) total_turnaround_time / n;

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

int main() {
        int n;

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

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

        // Input process details (burst time and priority)
        for (int i = 0; i < n; i++) {
        printf("\nEnter name of process %d: ", i + 1);
        scanf("%d", &processes[i]);
        printf("Enter burst time for process %d: ", processes[i]);
        scanf("%d", &burst_time[i]);
        printf("Enter priority for process %d: ", processes[i]);
        scanf("%d", &priority[i]);
        }

        // Calculate waiting time
        calculateWaitingTime(processes, n, burst_time, waiting_time, priority);

        // Calculate turnaround time
        calculateTurnaroundTime(burst_time, waiting_time, n, turnaround_time);
```

```
    // Display the results
    displayResults(processes, burst_time, priority, waiting_time, turnaround_time, n);

    return 0;
}
```

Output:




Program code:
( Preemptive )

```
#include <stdio.h>

// Structure to store process details
struct Process {
    int id, arrival_time, burst_time, priority, remaining_time, completion_time;
};

// Function to find the process with the highest priority at a given time
int getHighestPriorityProcess(struct Process proc[], int n, int current_time) {
    int highest_priority_index = -1;
    int highest_priority = 9999; // A large number to ensure lowest priority selection

    for (int i = 0; i < n; i++) {
    if (proc[i].arrival_time <= current_time && proc[i].remaining_time > 0) {
    if (proc[i].priority < highest_priority) {
        highest_priority = proc[i].priority;
        highest_priority_index = i;
    }
    }
    }

    return highest_priority_index;
}

// Function to implement Preemptive Priority Scheduling
void preemptivePriorityScheduling(struct Process proc[], int n) {
    int current_time = 0, completed = 0;
    int waiting_time[n], turnaround_time[n];

    // Initialize remaining time for each process
    for (int i = 0; i < n; i++) {
    proc[i].remaining_time = proc[i].burst_time;
    }

    while (completed < n) {
    int highest_priority_index = getHighestPriorityProcess(proc, n, current_time);
```

```
        if (highest_priority_index == -1) {
        current_time++; // If no process is available, move time forward
        continue;
        }

        // Process execution for 1 time unit
        proc[highest_priority_index].remaining_time--;
        current_time++;

        // If process is completed
        if (proc[highest_priority_index].remaining_time == 0) {
        proc[highest_priority_index].completion_time = current_time;
        completed++;
        }
        }

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

        // Print results
        printf("\nProcess\tAT\tBT\tPriority\tCT\tTAT\tWT\n");
        for (int i = 0; i < n; i++) {
        printf("%d\t%d\t%d\t%d\t\t%d\t%d\t%d\n", proc[i].id, proc[i].arrival_time,
proc[i].burst_time,
        proc[i].priority, proc[i].completion_time, turnaround_time[i], waiting_time[i]);
        }

        // Calculate average waiting time and turnaround time
        float avg_waiting_time = 0, avg_turnaround_time = 0;
        for (int i = 0; i < n; i++) {
        avg_waiting_time += waiting_time[i];
        avg_turnaround_time += turnaround_time[i];
        }
        avg_waiting_time /= n;
```

```
    avg_turnaround_time /= n;

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

int main() {
    int n;

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

    struct Process proc[n];

    // Input process details (arrival time, burst time, and priority)
    for (int i = 0; i < n; i++) {
    proc[i].id = i + 1;
    printf("\nEnter arrival time for process %d: ", proc[i].id);
    scanf("%d", &proc[i].arrival_time);
    printf("Enter burst time for process %d: ", proc[i].id);
    scanf("%d", &proc[i].burst_time);
    printf("Enter priority for process %d: ", proc[i].id);
    scanf("%d", &proc[i].priority);
    }

    // Execute Preemptive Priority Scheduling
    preemptivePriorityScheduling(proc, n);

    return 0;
}
```

Output:

```
┌──(student㉿kali)-[~]
└─$ nano priority_scheduling.c

┌──(student㉿kali)-[~]
└─$ gcc priority_scheduling.c -o priority_scheduling

┌──(student㉿kali)-[~]
└─$ ./priority_scheduling
Enter the number of processes: 3

Enter arrival time for process 1: 0
Enter burst time for process 1: 6
Enter priority for process 1: 2

Enter arrival time for process 2: 2
Enter burst time for process 2: 8
Enter priority for process 2: 1

Enter arrival time for process 3: 4
Enter burst time for process 3: 7
Enter priority for process 3: 3

Process AT      BT      Priority        CT      TAT     WT
1       0       6       2               14      14      8
2       2       8       1               10      8       0
3       4       7       3               21      17      10

Average Waiting Time: 6.00
Average Turnaround Time: 13.00
```

Result:

Hence the priority scheduling technique is implemented