Week-9

Write a C program to simulate the following CPU scheduling algorithm to find turnaround time and waiting time.

i) SJF (preemptive)
ii) Priority (preemptive and non-preemptive)
iii) Round Robin

```
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
#define MAX_PROCESS 10
#define MAX_NAME 5

typedef struct {
    char name[MAX_NAME];
    int arrivalTime;
    int burstTime;
    int priority;
    int remainingTime;
    int startTime;
    int waitingTime;
    int turnaroundTime;
    bool executed;
} Process;

Process processes[MAX_PROCESS];
void inputProcesses(int n){
    printf("Enter process details(name
        arrival_time burst_time priority):\n");
    for(int i=0; i<(n);i++){
        scanf("%s%d%d", processes[i].name,
            &processes[i].arrivalTime,
        &processes[i].burstTime, &processes[i].pri
```

```
        processes[i].remainingTime = processes[i].
        processes[i].executed = false;          burstTime;
    }
}
void sortProcessesByPriority(int n){
    for(int i=0; i<n-1; i++){
        for(int j=0; j<n-i-1; j++){
            if(processes[j].priority >
            processes[j+1].priority){
                Process temp = processes[j];
                processes[j] = processes[j+1];
                processes[j+1] = temp;
            }
        }
    }
}
void nonPreemptivePriority(int n){
    int currentTime = 0;
    for(int i=0; i<n; i++){
        sortProcessesByPriority(n);
        int shortest = -1;
        for(int j=0; j<n; j++){
            if(!processes[j].executed &&
            processes[j].arrivalTime <= currentTime)
            {
                shortest = j;
                break;
            }
    }
```

```
if (shortest == -1) {
    currentTime++;
    i--;
    continue;
}
processes[shortest].waitingTime = currentTime -
                    processes[shortest].arrivalTime;
processes[shortest].turnaroundTime =
                    processes[shortest].waitingTime
                  + processes[shortest].burstTime;
currentTime += processes[shortest].
                    burstTime;
processes[shortest].executed = true;
}
}
void preemptivePriority(int n){
int currentTime = 0;
int completed = 0;
while(completed < n){
sortProcessesByPriority(n);
int shortest = -1;
for(int i=0; i<n; i++){
if(!processes[i].executed &&
processes[i].arrivalTime <= currentTime){
if(shortest == -1 || processes[i].
priority < processes[shortest].priority){
shortest = i;
}
}
}
```

```
if(shortest == -1){
    currentTime++;
    continue;
}
processes[shortest].remainingTime--;
if(processes[shortest].remainingTime == 0){
    processes[shortest].waitingTime =
    currentTime + 1 - processes[shortest].arrivalTime -
        processes[shortest].burstTime;
    processes[shortest].turnaroundTime =
    processes[shortest].waitingTime +
        processes[shortest].burstTime;
    processes[shortest].executed = true;
    completed++;
}
currentTime++;
}
}
void preemptiveSJF(int n){
    int currentTime = 0;
    int completed = 0;
    while(completed < n){
        int shortest = -1;
        for(int i=0; i<n; i++){
            if(!processes[i].executed &&
            processes[i].arrivalTime <= currentTime){
                if(shortest == -1 || processes[i].
                remainingTime < processes[shortest].
                remainingTime){
                    shortest = i;
```

```
if (shortest == -1){
    currentTime++;
    continue;
}
processes[shortest].remainingTime--;
if(processes[shortest].remainingTime == 0){
    processes[shortest].waitingTime = currentTime +
    1 - processes[shortest].arrivalTime -
        processes[shortest].burstTime;
    processes[shortest].turnaroundTime =
    processes[shortest].waitingTime + processes[
    shortest].burstTime;
    processes[shortest].turnaroundTime =
    processes[shortest].waitingTime +
                    processes[shortest].burstTime
    processes[shortest].executed = true;
    completed++;
}
currentTime++;
}
void roundRobin(int n, int quantum) {
    int currentTime = 0;
    int completed = 0;
    int remaining[MAX_PROCESSES];
    for(int i = 0; i<n; i++){
        remaining[i] = processes[i].burstTime;
    }
    while(completed < n){
        for(int i=0; i<n; i++){
            if(processes[i].arrivalTime <=
            currentTime && remaining[i] > 0){
```

```
if (remaining[p] > quantum) {
    currentTime += quantum;
    remaining[i] -= quantum;
} else {
    currentTime += remaining[i];
    processes[i].waitingTime =
    currentTime - processes[i].arrivalTime
    - processes[i].burstTime;
processes[i].turnaroundTime = processes[i].
waitingTime + processes[p].burstTime;
remaining[p]=0;
completed++;
}
}
}
}
void displayResults(int n){
printf("Process\t Turnaround Time\t
    Waiting Time\n");
for(int i=0; i<n; i++){
printf("%d\t%d\t\t%d\n",
processes[i].name, processes[i].turnaround
                    Time,
    processes[i].waitingTime);
}
}
```

```
float calculateAverage(int n, char type){
    float total = 0;
    for(int i=0; i<n; i++){
        if(type == 'W')
            total += process[i].waitingTime;
        else if (type == 'T')
            total += process[i].turnaroundTime;
    }
    return total/n;
}
int main(){
    int n, quantum;
    printf("Enter the number of processes:");
    scanf("%d", &n);
    if (n > MAX_PROCESS){
        printf("Number of processes exceeds
        the limit\n");
        return 0;
    }
    inputProcesses(n);
    printf("Enter quantum for Round Robin:");
    scanf("%d", &quantum);
    printf("\n Non-preemptive priority
        scheduling:\n");
    nonPreemptivePriority(n);
    displayResults(n);
    printf("Average waiting time: %0.2f\n",
        calculateAverage(n,'W'));
```

```
printf("Average turnaround time: %.2f\n",
        calculateAverage(n, 'T'));

printf("\n Preemptive priority scheduling:
\n");
for(int i=0; i<(n);i++){
    processes[i].executed=false;
    processes[i].remainingTime = processes[i].
                                 burstTime;
}
preemptivePriority(n);
displayResults(n);
printf("Average waiting Time: %.2f\n",
    calculateAverage(n, 'W'));
printf("Average Turnaround time: %.2f\n",
    calculateAverage(n, 'T'));
printf("\n Preemptive SJF scheduling\n");
for(int i=0; i<n; i++){
    processes[i].executed=false;
    processes[i].remainingTime= processes[i].
                                burstTime;
}
preemptiveSJF(n);
displayResults(n);
printf("Average waiting time: %.2f\n",
        calculateAverage(n,'W'));
printf("Average Turnaround Time: %.2f
        \n", calculateAverage
        (n,'T'));
printf("\n\nRound Robin Scheduling (Quantum=
    %d): \n", quantum);
for(int i=0; i<n;i++){
    processes[i].executed=false;
```

processes[i].remainingTime = processes[i].
burstTime;

}

roundRobin(n, quantum);

displayResults(n);

printf("Average Waiting Time: %.2f\n",
calculateAverage(n, 'W'));

printf("Average Turnaround Time: %.2f\n",
calculateAverage(n, 'T'));

return 0;

}

Output: Enter the no. of processes: 3

1
4
6
1

2
2
8
2

3
7
3
3

No. of quantums: 2

Non-Preemptive Priority Scheduling:

Processes	Turnaround Time	Waiting Time
1	12	6
2	8	0
3	12	9

Average waiting time: 5.00

Average turnaround time: 10.67

Preemptive Priority Scheduling:

Process	Turnaround Time	Waiting Time
1	6	0
2	14	6
3	12	9

Average Waiting Time: 5.00
Average Turnaround Time: 10.67

Preemptive SJF Scheduling:

Process	Turnaround Time	Waiting Time
1	6	0
2	17	9
3	6	3

Average Waiting Time: 4.00
Average Turnaround Time: 9.67

Round Robin Scheduling (Quantum = 2):

Process	Turnaround Time	Waiting Time
1	12	7
2	17	9
3	6	3

Average Waiting Time: 6.34
Average Turnaround Time: 12.00

31/5/24