SJF (Preemptive) & Priority (Pre-emptive & Non-Pre-emptive)

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 details:");
    for(int i=0; i<n; i++) {
        scanf("%s %d %d %d", processes[i].name,
              &processes[i].arrivalTime,
              &processes[i].burstTime, &processes[i].priority);
        processes[i].remainingTime = processes[i].burstTime;
        processes[i].executed = false;
    }
}
```

```
void sortProcessesByPriority(int n) {
    for (int i=0; i<n-1; i++) {
        for (int j=0; j<n-1; j++) {
            if (process[j].priority > process[j+1].
                                        priority) {
                process temp = process[j];
                process[j] = process[j+1];
                process[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 (!process[j].executed &&
                process[j].arrivalTime <= currentTime) {
                shortest = j;
                break;
            }
        }
        if (shortest == -1) {
            currentTime++;
            i--;
            continue;
        }
        process[shortest].waitingTime = currentTime -
                                process[shortest].arrivalTime;
        process[shortest].turnaroundTime =
        process[shortest].waitingTime + process[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].executed = true;
```

```
            completed++;
        }
        currentTime++;
    }
}
void roundRobin(int n, int quantum) {
    int currentTime = 0;
    int completed = 0;
    int remaining[MAX_PROCESS];
    for(int i=0; i<n; i++) {
        remaining[i] = process[i].burstTime;
    }
    while(completed < n) {
        for(int i=0; i<n; i++) {
            if(process[i].arrivalTime <= currentTime
            && remaining > 0) {
                if(remaining[i] > quantum) {
                    currentTime += quantum;
                    remaining[i] -= quantum;
                }
                else {
                    currentTime += remaining[i];
                    process[i].waitingTime =
                    currentTime - process[i].
                    arrivalTime - process[i].burstTime;
                    process[i].turnaroundTime =
                    process[i].waitingTime +
                        process[i].burstTime;
                    remaining[i] = 0;
                    completed++;
                }
            }
        }
    }
}
```

```
void displayResults(int n) {
    printf("Process\tTurnaround Time\tWaiting Time\n");
    for(int i=0; i<n; i++) {
        printf("%d\t%d\t%d\n", process[i].pno,
        process[i].turnaroundTime, process[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 no. of processes: ");
    scanf("%d", &n);
    if(n > MAX_PROCESS) {
        printf("Limit exceeded");
        return 1;
    }
    inputProcesses(n);
    printf("no. of quantum: ");
    scanf("%d", &quantum);
```

```
printf("\n Non-Preemptive Priority Scheduling:\n");
nonPreemptive(n);
displayResults(n);
printf("Average Waiting Time: %.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++) {
    process[i].executed = false;
    process[i].remainingTime = process[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("Preemptive SJF Scheduling\n");
for(int i=0; i<n; i++) {
    process[i].executed = false;
    process[i].remainingTime = process[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("Round Robbin Scheduling (Quantum = %d): \n",
                                        quantum);
for(int i=0; i<n; i++){
    process[i].executed = false;
    process[i].remainingTime = process[i].burstTime;
}
roundRobin(n, quantum);
displayResult(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

No. of quantums: 2

Non-Preemptive Priority Scheduling:

Process	Turnaround Time	Waiting Time
1	12	6
2	8	0
3	12	0

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	Waiting Time
1	8	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: ~~11.00~~ 6.34
Average Turnaround Time: 12.00

Sm
3/6/24.