NAME - SHREERAM
REG. NO. - 20214033

Question 01:

Write a C program to simulate the following CPU scheduling algorithms to find turnaround time and waiting time for the given problem. **Assumption:** all the processes arrive at the same time, time slice t=2 sec (for RR scheduling), priority assigned to the processes as P2>P3>P1>P4 (For priority scheduling).

a) First come first serve (FCFS)
b) Shortest Job First (SJF)
c) Round Robin (RR)
d) Priority

Processes	Burst Time
P1	24
P2	3
P3	3
P4	7

A. FCFS-> Code:

```c
#include <stdio.h>

int max(int a,int b){
    if(a>b)return a;
    return b;
}
void findWaitingTime(int tat[], int n, int bt[], int wt[]) {
    int i;
    for(i = 0; i < n; i++){
        wt[i] = tat[i] - bt[i];
    }
}

void findCompletionTime(int arrival[], int n, int bt[], int completiontime[]) {
    completiontime[0] = arrival[0]+bt[0];
    int i;
    for(i = 1; i < n; i++){
        completiontime[i] = bt[i] + max(completiontime[i-1],arrival[i]);
    }
}
```

```c
void findTurnAroundTime(int arrival[], int n, int com[], int tat[]) {
    int i;
    for(i = 0; i < n; i++){
        tat[i] = com[i] - arrival[i];
    }
}

int main() {
    int n ;
    printf("Enter Number of Processes : ");scanf("%d",&n);
    int arrival[n], burst_time[n];int i;
    printf("Enter Arrival and Burst time for processes
\n");for(i=0;i<n;i++){printf("%d ",i);scanf("%d %d",&arrival[i],&burst_time[i]);}
    int waiting_time[n], turnaround_time[n],completiontime[n];
     findCompletionTime(arrival, n, burst_time, completiontime);
    findTurnAroundTime(arrival, n,completiontime,turnaround_time);
    findWaitingTime(turnaround_time, n, burst_time, waiting_time);

    printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for(i = 0; i < n; i++){
        printf("P%d\t%d\t\t%d\t\t%d\n", i , burst_time[i], waiting_time[i],
turnaround_time[i]);
    }

    return 0;
}
```

B. SJF -> Code:

```c
#include <stdio.h>
#include <stdlib.h>

struct Process {
    int processID;
    int burstTime;
};

int compare(const void *a, const void *b) {
    return ((struct Process *)a)->burstTime - ((struct Process *)b)->burstTime;
}

void findWaitingTime(struct Process proc[], int n, int wt[]) {
```

```
    wt[0] = 0;
    for (int i = 1; i < n; i++) {
        wt[i] = proc[i - 1].burstTime + wt[i - 1];
    }
}

void findTurnAroundTime(struct Process proc[], int n, int wt[], int tat[]) {
    for (int i = 0; i < n; i++) {
        tat[i] = proc[i].burstTime + wt[i];
    }
}

int main() {
    struct Process proc[] = {{1, 24}, {2, 3}, {3, 3}, {4, 7}};
    int n = sizeof(proc) / sizeof(proc[0]);
    int waiting_time[n], turnaround_time[n];

    qsort(proc, n, sizeof(proc[0]), compare);
    findWaitingTime(proc, n, waiting_time);
    findTurnAroundTime(proc, n, waiting_time, turnaround_time);

    printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++) {
        printf("P%d\t%d\t\t%d\t\t%d\n", proc[i].processID, proc[i].burstTime,
waiting_time[i], turnaround_time[i]);
    }

    return 0;
}
```

C. RR -> Code:

```
 #include <stdio.h>


void findWaitingTime(int processes[], int n, int bt[], int wt[], int quantum) {
    int remaining_time[n];
    for (int i = 0; i < n; i++) {
        remaining_time[i] = bt[i];
        wt[i] = 0;
    }

    int t = 0;
```

```c
    while (1) {
        int done = 1;
        for (int i = 0; i < n; i++) {
            if (remaining_time[i] > 0) {
                done = 0;
                if (remaining_time[i] > quantum) {
                    t += quantum;
                    remaining_time[i] -= quantum;
                } else {
                    t = t + remaining_time[i];
                    wt[i] = t - bt[i];
                    remaining_time[i] = 0;
                }
            }
        }
        if (done == 1)
            break;
    }
}

void findTurnAroundTime(int processes[], int n, int bt[], int wt[], int tat[]) {
    for (int i = 0; i < n; i++) {
        tat[i] = bt[i] + wt[i];
    }
}

int main() {
    int processes[] = {1, 2, 3, 4};
    int n = sizeof(processes) / sizeof(processes[0]);
    int burst_time[] = {24, 3, 3, 7};
    int quantum = 2;
    int waiting_time[n], turnaround_time[n];

    findWaitingTime(processes, n, burst_time, waiting_time, quantum);
    findTurnAroundTime(processes, n, burst_time, waiting_time, turnaround_time);

    printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++) {
        printf("P%d\t%d\t\t%d\t\t%d\n", processes[i], burst_time[i],
waiting_time[i], turnaround_time[i]);
    }

    return 0;
}
```

D. Priority -> Code:

```c
#include <stdio.h>
#include <stdlib.h>
struct Process {
    int processID;
    int burstTime;
    int priority;
};
int compare(const void *a, const void *b) {
    return ((struct Process *)a)->priority - ((struct Process *)b)->priority;
}

void findWaitingTime(struct Process proc[], int n, int wt[]) {
    wt[0] = 0;
    for (int i = 1; i < n; i++) {
        wt[i] = proc[i - 1].burstTime + wt[i - 1];
    }
}

void findTurnAroundTime(struct Process proc[], int n, int wt[], int tat[]) {
    for (int i = 0; i < n; i++) {
        tat[i] = proc[i].burstTime + wt[i];
    }
}

int main() {
    struct Process proc[] = {{1, 24, 2}, {2, 3, 1}, {3, 3, 4}, {4, 7, 3}};
    int n = sizeof(proc) / sizeof(proc[0]);
    int waiting_time[n], turnaround_time[n];
    qsort(proc, n, sizeof(proc[0]), compare);
    findWaitingTime(proc, n, waiting_time);
    findTurnAroundTime(proc, n, waiting_time, turnaround_time);

    printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++) {
        printf("P%d\t%d\t\t%d\t\t%d\n", proc[i].processID, proc[i].burstTime,
waiting_time[i], turnaround_time[i]);
    }
    return 0;
}
```

Question 02:

Repeat the problem 1 with an assumption that processes arrived as per the given arrival times:

Processes	Burst Time	Arrival Time
P1	24	0
P2	3	2
P3	3	1
P4	7	3

A. FCFS-> Code:

```c
#include <stdio.h>

// #include<maths.h>
int max(int a,int b){
    if(a>b)return a;
    return b;
}
void findWaitingTime(int tat[], int n, int bt[], int wt[]) {
    int i;
    for(i = 0; i < n; i++){
        wt[i] = tat[i] - bt[i];
    }
}

void findCompletionTime(int arrival[], int n, int bt[], int completiontime[]) {
    completiontime[0] = arrival[0]+bt[0];
    int i;
    for(i = 1; i < n; i++){
        completiontime[i] = bt[i] + max(completiontime[i-1],arrival[i]);
    }
}


void findTurnAroundTime(int arrival[], int n, int com[], int tat[]) {
    int i;
    for(i = 0; i < n; i++){
        tat[i] = com[i] - arrival[i];
    }
}
```

```c
int main() {
    int n ;
    printf("Enter Number of Processes : ");scanf("%d",&n);
    int arrival[n], burst_time[n];int i;
    printf("Enter Arrival and Burst time for processes
\n");for(i=0;i<n;i++){printf("%d ",i);scanf("%d %d",&arrival[i],&burst_time[i]);}
    int waiting_time[n], turnaround_time[n],completiontime[n];
     findCompletionTime(arrival, n, burst_time, completiontime);
    findTurnAroundTime(arrival, n,completiontime,turnaround_time);
    findWaitingTime(turnaround_time, n, burst_time, waiting_time);

    printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for(i = 0; i < n; i++){
        printf("P%d\t%d\t\t%d\t\t%d\n", i , burst_time[i], waiting_time[i],
turnaround_time[i]);
    }

    return 0;
}
```

B. SJF-> Code:

```c
     #include <stdio.h>
#include <stdlib.h>

struct Process {
    int processID;
    int burstTime;
    int arrivalTime;
};

int compareArrival(const void *a, const void *b) {
    return ((struct Process *)a)->arrivalTime - ((struct Process *)b)-
>arrivalTime;
}

int compareBurst(const void *a, const void *b) {
    return ((struct Process *)a)->burstTime - ((struct Process *)b)->burstTime;
}

void findWaitingTime(struct Process proc[], int n, int wt[]) {
    int service_time[n];
```

```
    service_time[0] = proc[0].arrivalTime;
    wt[0] = 0;

    for (int i = 1; i < n; i++) {
        service_time[i] = service_time[i - 1] + proc[i - 1].burstTime;
        wt[i] = service_time[i] - proc[i].arrivalTime;
        if (wt[i] < 0) wt[i] = 0;
    }
}

void findTurnAroundTime(struct Process proc[], int n, int wt[], int tat[]) {
    for (int i = 0; i < n; i++) {
        tat[i] = proc[i].burstTime + wt[i];
    }
}

int main() {
    struct Process proc[] = {{1, 24, 0}, {2, 3, 2}, {3, 3, 1}, {4, 7, 3}};
    int n = sizeof(proc) / sizeof(proc[0]);
    int waiting_time[n], turnaround_time[n];

    qsort(proc, n, sizeof(proc[0]), compareArrival);
    findWaitingTime(proc, n, waiting_time);
    qsort(proc, n, sizeof(proc[0]), compareBurst);
    findTurnAroundTime(proc, n, waiting_time, turnaround_time);

    printf("Process\tBurst Time\tArrival Time\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++) {
        printf("P%d\t%d\t\t%d\t\t%d\t\t%d\n", proc[i].processID,
proc[i].burstTime, proc[i].arrivalTime, waiting_time[i], turnaround_time[i]);
    }

    return 0;
}
```

C. RR-> Code:

```
 #include <stdio.h>


void findWaitingTime(int processes[], int n, int bt[], int wt[], int at[], int
quantum) {
    int remaining_time[n];
```

```
    for (int i = 0; i < n; i++) {
        remaining_time[i] = bt[i];
        wt[i] = 0;
    }

    int t = 0;
    int flag = 0;
    while (1) {
        flag = 1;
        for (int i = 0; i < n; i++) {
            if (remaining_time[i] > 0 && at[i] <= t) {
                flag = 0;
                if (remaining_time[i] > quantum) {
                    t += quantum;
                    remaining_time[i] -= quantum;
                } else {
                    t = t + remaining_time[i];
                    wt[i] = t - bt[i] - at[i];
                    remaining_time[i] = 0;
                }
            }
        }
        if (flag == 1)
            break;
    }
}

void findTurnAroundTime(int processes[], int n, int bt[], int wt[], int tat[]) {
    for (int i = 0; i < n; i++) {
        tat[i] = bt[i] + wt[i];
    }
}

int main() {
    int processes[] = {1, 2, 3, 4};
    int n = sizeof(processes) / sizeof(processes[0]);
    int burst_time[] = {24, 3, 3, 7};
    int arrival_time[] = {0, 2, 1, 3};
    int quantum = 2;
    int waiting_time[n], turnaround_time[n];

    findWaitingTime(processes, n, burst_time, waiting_time, arrival_time,
quantum);
    findTurnAroundTime(processes, n, burst_time, waiting_time, turnaround_time);
```

```
    printf("Process\tBurst Time\tArrival Time\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++) {
        printf("P%d\t%d\t\t%d\t\t%d\t\t%d\n", processes[i], burst_time[i],
arrival_time[i], waiting_time[i], turnaround_time[i]);
    }

    return 0;
}
```

D. priority-> Code:

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

struct Process {
    int processID;
    int burstTime;
    int arrivalTime;
    int priority;
};

int compareArrival(const void *a, const void *b) {
    return ((struct Process *)a)->arrivalTime - ((struct Process *)b)-
>arrivalTime;
}

int comparePriority(const void *a, const void *b) {
    return ((struct Process *)a)->priority - ((struct Process *)b)->priority;
}

void findWaitingTime(struct Process proc[], int n, int wt[]) {
    int service_time[n];
    service_time[0] = proc[0].arrivalTime;
    wt[0] = 0;

    for (int i = 1; i < n; i++) {
        service_time[i] = service_time[i - 1] + proc[i - 1].burstTime;
        wt[i] = service_time[i] - proc[i].arrivalTime;
        if (wt[i] < 0) wt[i] = 0;
    }
}

void findTurnAroundTime(struct Process proc[], int n, int wt[], int tat[]) {
```

```
    for (int i = 0; i < n; i++) {
        tat[i] = proc[i].burstTime + wt[i];
    }
}

int main() {
    struct Process proc[] = {{1, 24, 0, 2}, {2, 3, 2, 1}, {3, 3, 1, 4}, {4, 7, 3,
3}};
    int n = sizeof(proc) / sizeof(proc[0]);
    int waiting_time[n], turnaround_time[n];

    qsort(proc, n, sizeof(proc[0]), compareArrival);
    findWaitingTime(proc, n, waiting_time);
    qsort(proc, n, sizeof(proc[0]), comparePriority);
    findTurnAroundTime(proc, n, waiting_time, turnaround_time);

    printf("Process\tBurst Time\tArrival Time\tPriority\tWaiting Time\tTurnaround
Time\n");
    for (int i = 0; i < n; i++) {
        printf("P%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", proc[i].processID,
proc[i].burstTime, proc[i].arrivalTime, proc[i].priority, waiting_time[i],
turnaround_time[i]);
    }

    return 0;
}
```

Question 03:

For RR Scheduling in problem 1 & 2: Vary the time slice/quantum of RR scheduling from 1 to 3sec(in steps of 1 sec) and plot a graph showing how the average turnaround time for processes vary with time slice/quantum. Also, plot a graph showing how the average waiting time for processes varies with time slice/quantum.

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

```c
void findWaitingTime(int processes[], int n, int bt[], int wt[], int at[], int
quantum) {
    int remaining_time[n];
    for (int i = 0; i < n; i++) {
        remaining_time[i] = bt[i];
        wt[i] = 0;
    }

    int t = 0;
    int flag = 0;
    while (1) {
        flag = 1;
        for (int i = 0; i < n; i++) {
            if (remaining_time[i] > 0 && at[i] <= t) {
                flag = 0;
                if (remaining_time[i] > quantum) {
                    t += quantum;
                    remaining_time[i] -= quantum;
                } else {
                    t = t + remaining_time[i];
                    wt[i] = t - bt[i] - at[i];
                    remaining_time[i] = 0;
                }
            }
        }
        if (flag == 1)
            break;
    }
}

void findTurnAroundTime(int processes[], int n, int bt[], int wt[], int tat[]) {
    for (int i = 0; i < n; i++) {
        tat[i] = bt[i] + wt[i];
    }
}

int main() {
    int processes[] = {1, 2, 3, 4};
    int n = sizeof(processes) / sizeof(processes[0]);
    int burst_time[] = {24, 3, 3, 7};
    int arrival_time[] = {0, 2, 1, 3};
    int waiting_time[n], turnaround_time[n];

    FILE *turnaround_file = fopen("turnaround_data.dat", "w");
    FILE *waiting_file = fopen("waiting_data.dat", "w");
```

```
    printf("Quantum\tAverage Turnaround Time\tAverage Waiting Time\n");
    for (int quantum = 1; quantum <= 3; quantum++) {
        float total_turnaround = 0, total_waiting = 0;
        for (int i = 0; i < n; i++) {
            waiting_time[i] = 0;
            turnaround_time[i] = 0;
        }

        findWaitingTime(processes, n, burst_time, waiting_time, arrival_time,
quantum);
        findTurnAroundTime(processes, n, burst_time, waiting_time,
turnaround_time);

        for (int i = 0; i < n; i++) {
            total_turnaround += turnaround_time[i];
            total_waiting += waiting_time[i];
        }

        float avg_turnaround = total_turnaround / n;
        float avg_waiting = total_waiting / n;

        fprintf(turnaround_file, "%d %f\n", quantum, avg_turnaround);
        fprintf(waiting_file, "%d %f\n", quantum, avg_waiting);

        printf("%d\t%f\t%f\n", quantum, avg_turnaround, avg_waiting);
    }

    fclose(turnaround_file);
    fclose(waiting_file);

    // Plotting
    FILE *gnuplotPipe = popen("gnuplot -persistent", "w");
    fprintf(gnuplotPipe, "set title 'Round Robin Scheduling'\n");
    fprintf(gnuplotPipe, "set xlabel 'Quantum'\n");
    fprintf(gnuplotPipe, "set ylabel 'Time'\n");
    fprintf(gnuplotPipe, "plot 'turnaround_data.dat' with linespoints title
'Average Turnaround Time', \
            'waiting_data.dat' with linespoints title 'Average Waiting Time'\n");
    fclose(gnuplotPipe);

    return 0;
}
```