Program 2. Simulate the following CPU scheduling algorithms to find turnaround time and waiting time a) FCFS b) SJF c) Round Robin d) Priority.

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

// Process structure
typedef struct {
    int pid;
    int burstTime;
    int waitingTime;
    int turnaroundTime;
    int arrivalTime; // Assuming all processes arrive at time 0
    int priority;
    int remainingTime; // For RR and SJF
} Process;

// Function Prototypes
void FCFS(Process p[], int n);
void SJF(Process p[], int n);
void RoundRobin(Process p[], int n, int quantum);
void PriorityScheduling(Process p[], int n);
void calculateMetrics(Process p[], int n);
void resetTimes(Process p[], int n);

// Main function
int main() {
    int quantum = 4; // Quantum time for RR
    // Process array
    Process proc[] = {{1, 6, 0, 0, 0, 2}, {2, 8, 0, 0, 0, 1}, {3, 7, 0, 0, 0, 3}, {4, 3, 0, 0, 0,
4}};
    int n = sizeof(proc) / sizeof(proc[0]);

    // FCFS Scheduling
    printf("First-Come, First-Served Scheduling\n");
    FCFS(proc, n);
    calculateMetrics(proc, n);
    resetTimes(proc, n);
```

```
// SJF Scheduling
    printf("Shortest Job First Scheduling\n");
    SJF(proc, n);
    calculateMetrics(proc, n);
    resetTimes(proc, n);

    // Round Robin Scheduling
    printf("Round Robin Scheduling\n");
    RoundRobin(proc, n, quantum);
    calculateMetrics(proc, n);
    resetTimes(proc, n);

    // Priority Scheduling
    printf("Priority Scheduling\n");
    PriorityScheduling(proc, n);
    calculateMetrics(proc, n);
    resetTimes(proc, n);

    return 0;
}

// FCFS Scheduler
void FCFS(Process p[], int n) {
    int currentTime = 0;
    for (int i = 0; i < n; i++) {
        p[i].waitingTime = currentTime;
        currentTime += p[i].burstTime;
        p[i].turnaroundTime = currentTime;
    }
}

// SJF Scheduler
void SJF(Process p[], int n) {
    int completed = 0, currentTime = 0, shortest = 0;
    int minm = INT_MAX;
    bool found = false;

    for (int i = 0; i < n; i++) {
        p[i].remainingTime = p[i].burstTime;
    }
```

```
    while (completed != n) {
        // Find shortest remaining time among the processes that have arrived by
currentTime
        minm = INT_MAX;
        shortest = -1;
        found = false;
        for (int i = 0; i < n; i++) {
            if ((p[i].arrivalTime <= currentTime) && (p[i].remainingTime < minm) &&
p[i].remainingTime > 0) {
                minm = p[i].remainingTime;
                shortest = i;
                found = true;
            }
        }

        // If no process was found, increment currentTime
        if (!found) {
            currentTime++;
            continue;
        }

        // Reduce remaining time by 1 since it's a unit time cycle
        p[shortest].remainingTime--;
        minm = p[shortest].remainingTime;
        if (minm == 0) minm = INT_MAX;

        // If a process gets completely executed
        if (p[shortest].remainingTime == 0) {
            completed++;
            p[shortest].turnaroundTime = currentTime + 1 - p[shortest].arrivalTime;
            p[shortest].waitingTime = p[shortest].turnaroundTime - p[shortest].burstTime;

            if (p[shortest].waitingTime < 0) {
                p[shortest].waitingTime = 0;
            }
        }
        currentTime++;
    }
}
```

```
// Round Robin Scheduler
void RoundRobin(Process p[], int n, int quantum) {
    int currentTime = 0;
    int remain = n;

    for (int i = 0; i < n; i++) {
        p[i].remainingTime = p[i].burstTime;
    }

    while (remain != 0) {
        for (int i = 0; i < n; i++) {
            if (p[i].remainingTime <= quantum && p[i].remainingTime > 0) {
                currentTime += p[i].remainingTime;
                p[i].remainingTime = 0;
                remain--;
                p[i].turnaroundTime = currentTime;
                p[i].waitingTime = currentTime - p[i].burstTime;
            } else if (p[i].remainingTime > 0) {
                p[i].remainingTime -= quantum;
                currentTime += quantum;
            }
        }
    }
}

// Priority Scheduler
void PriorityScheduling(Process p[], int n) {
    // Sorting processes based on priority using a simple bubble sort
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (p[j].priority > p[j + 1].priority) {
                Process temp = p[j];
                p[j] = p[j + 1];
                p[j + 1] = temp;
            }
        }
    }
    // Scheduling as FCFS after sorting based on priority
    FCFS(p, n);
}

// Calculate and print metrics
void calculateMetrics(Process p[], int n) {
    float totalWaitingTime = 0, totalTurnaroundTime = 0;
```

```
    // Printing table header
    printf("\n+---------+--------------+-----------------+\n");
    printf("| Process | Waiting Time  | Turnaround Time |\n");
    printf("+---------+--------------+-----------------+\n");

    for (int i = 0; i < n; i++) {
        totalWaitingTime += p[i].waitingTime;
        totalTurnaroundTime += p[i].turnaroundTime;

        // Printing each process metrics
        printf("| P%-7d| %-13d| %-17d|\n", p[i].pid, p[i].waitingTime,
p[i].turnaroundTime);
    }

    // Printing table footer
    printf("+---------+--------------+-----------------+\n");

    // Printing average times
    printf("Average waiting time = %.2f\n", totalWaitingTime / n);
    printf("Average turnaround time = %.2f\n\n", totalTurnaroundTime / n);
}

// Reset times for the processes
void resetTimes(Process p[], int n) {
    for (int i = 0; i < n; i++) {
        p[i].waitingTime = 0;
        p[i].turnaroundTime = 0;
    }
}
```

```
gcc -o program 2.c -lpthread -lrt
./program
```