Objective: To design a CPU scheduler for simulating a few CPU Scheduling policies.

```
#include <stdio.h>
typedef struct {
    int pid;
    int arrival;
    int burst;
    int remaining;
    int start;
    int completion;
    int waiting;
    int turnaround;
    int response;
    int started;
} Process;

void fcfs(Process p[], int n) {
    int time = 0;
    float awt = 0, att = 0, art = 0;
    printf("\nGantt Chart:\n|");
    for (int i = 0; i < n; i++) {
        if (time < p[i].arrival)
            time = p[i].arrival;
        p[i].start = time;
        p[i].response = p[i].start - p[i].arrival;
        time += p[i].burst;
        p[i].completion = time;
        p[i].turnaround = p[i].completion - p[i].arrival;
        p[i].waiting = p[i].turnaround - p[i].burst;
        printf(" P%d |", p[i].pid);
        awt += p[i].waiting;
        att += p[i].turnaround;
        art += p[i].response;
    }

    printf("\nAverage Waiting Time = %.2f", awt / n);
    printf("\nAverage Turnaround Time = %.2f", att / n);
    printf("\nAverage Response Time = %.2f\n", art / n);
}

void roundRobin(Process p[], int n, int tq) {
    int time = 0, completed = 0;
    float awt = 0, att = 0, art = 0;
    printf("\nGantt Chart:\n|");

    while (completed < n) {
        for (int i = 0; i < n; i++) {
            if (p[i].arrival <= time && p[i].remaining > 0) {
                if (!p[i].started) {
                    p[i].start = time;
                    p[i].response = p[i].start - p[i].arrival;
                    p[i].started = 1;
                }
                printf(" P%d |", p[i].pid);

                if (p[i].remaining > tq) {
                    time += tq;
                    p[i].remaining -= tq;
                } else {
                    time += p[i].remaining;
                    p[i].remaining = 0;
                    p[i].completion = time;
                    p[i].turnaround = p[i].completion - p[i].arrival;
                    p[i].waiting = p[i].turnaround - p[i].burst;
                    completed++;

                    awt += p[i].waiting;
                    att += p[i].turnaround;
                    art += p[i].response;
                }
            }
        }
    }
    printf("\nAverage Waiting Time = %.2f", awt / n);
    printf("\nAverage Turnaround Time = %.2f", att / n);
    printf("\nAverage Response Time = %.2f\n", art / n);
}

int main() {
    int n = 5, choice, tq;
    Process p[5] = {{1,0,3,3,0,0,0,0,0,0},
                        {2,2,6,6,0,0,0,0,0,0},
                        {3,4,4,4,0,0,0,0,0,0},
                        {4,6,5,5,0,0,0,0,0,0},
                        {5,8,2,2,0,0,0,0,0,0}};
    do {
        printf("\n1. FCFS\n2. Round Robin\n5. Exit\nEnter choice: ");
        scanf("%d", &choice);
        switch (choice) {
        case 1:
            fcfs(p, n);
            break;
        case 2:
            printf("Enter Time Quantum: ");
            scanf("%d", &tq);
            roundRobin(p, n, tq);
            break;
        case 5:
            return 0;
        default:
            printf("Invalid choice");
        }

    } while (1);
}
```

Objective: Implementation of Banker’s algorithm to avoid deadlock

```
#include <stdio.h>

int main() {
    int alloc[5][4] = {{0,0,1,2},{2,0,0,0},{0,0,3,4},{2,3,5,4},{0,3,3,2}};
    int max[5][4] = {{0,0,1,2},{2,7,5,0}, {6,6,5,6},{4,3,5,6},{0,6,5,2}};
    int avail[4] = {0,1,0,4};
    int need[5][4], finish[5] = {0};

    for(int i=0;i<5;i++)
        for(int j=0;j<4;j++)
            need[i][j] = max[i][j] - alloc[i][j];

    printf("Need Matrix:\n");

    for(int i=0;i<5;i++) {
        for(int j=0;j<4;j++)
            printf("%d ", need[i][j]);
        printf("\n");
    }

    int safeSeq[5], count = 0;

    while(count < 5) {
        int found = 0;
        for(int i=0;i<5;i++) {
            if(!finish[i]) {
                int j;
                for(j=0;j<4;j++)
                    if(need[i][j] > avail[j])
                        break;
                if(j == 4) {
                    for(int k=0;k<4;k++)
                        avail[k] += alloc[i][k];
                    safeSeq[count++] = i;
                    finish[i] = 1;
                    found = 1;
                }
            }
        }
        if(!found) break;
    }

    if(count == 5) {
        printf("System is in safe state.\nSafe Sequence: ");
        for(int i=0;i<5;i++)
            printf("P%d ", safeSeq[i]+1);
    } else {
        printf("System is not in safe state.");
    }
    return 0;
}
```