Scheduling Algorithm:

1.Write a menu driven c program to implement various CPU scheduling algorithm (FIFO, SJF, Priority, Round robin) and calculate the waiting , turnaround time for each process and calculate average waiting and turnaround time.

```c
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#define MAX_PROCESSES 100
#define HIGH_PRIORITY 9999

void fifo(int n, int bt[], int at[]);
void sjf(int n, int bt[], int at[]);
void priority_scheduling(int n, int bt[], int at[], int priority[]);
void round_robin(int n, int bt[], int at[], int time_quantum);

void swap(int* a, int* b) {
int temp = *a;
*a = *b;
*b = temp;
}

int main() {
    while (1) {
        int n;
        int bt[MAX_PROCESSES];
        int at[MAX_PROCESSES];
        int priority[MAX_PROCESSES];
        int time_quantum;
        int choice;
        printf("\nChoose a scheduling algorithm:\n");
        printf("1. FIFO (First-In, First-Out)\n");
        printf("2. SJF (Shortest Job First)\n");
        printf("3. Priority Scheduling\n");
```

```
        printf("4. Round Robin\n");
        printf("5. Exit\n");
        scanf("%d", &choice);

        if (choice == 5) {
            printf("Exiting...\n");
            break;
        }

        printf("Enter the number of processes: ");
        scanf("%d", &n);

        printf("Enter the burst times of the processes:\n");
        for (int i = 0; i < n; i++) {
        printf("Process %d burst time: ", i + 1);
        scanf("%d", &bt[i]);
    }

    printf("Enter the arrival times of the processes:\n");
    for (int i = 0; i < n; i++) {
        printf("Process %d arrival time: ", i + 1);
        scanf("%d", &at[i]);
    }

    printf("Enter the priorities of the processes (1 = highest priority):\n");
    for (int i = 0; i < n; i++) {
        printf("Process %d priority: ", i + 1);
        scanf("%d", &priority[i]);
    }

    printf("Enter the time quantum for Round Robin: ");
    scanf("%d", &time_quantum);
    switch (choice) {
        case 1:
            fifo(n, bt, at);
            break;
        case 2:
            sjf(n, bt, at);
            break;
        case 3:
            priority_scheduling(n, bt, at,priority);
            break;
        case 4:
            round_robin(n, bt, at,time_quantum);
            break;
        default:
            printf("Invalid choice.\n");
        }
```

```
    }
    return 0;
    }
// FIFO scheduling
void fifo(int n, int bt[], int at[]) {
    int completion_time[n];
    int turnaround_time[n];
    int waiting_time[n];
    int current_time = 0;
    for (int i = 0; i < n; i++) {
        if (current_time < at[i]) {
            current_time = at[i];
        }
        current_time += bt[i];
        completion_time[i] = current_time;
        turnaround_time[i] = completion_time[i] - at[i];
        waiting_time[i] = turnaround_time[i] - bt[i];
        printf("Process %d - Turnaround Time: %d, Waiting Time: %d\n", i + 1,
turnaround_time[i], waiting_time[i]);
    }

    float total_turnaround = 0.0f;
    float total_waiting = 0.0f;
    for (int i = 0; i < n; i++) {
        total_turnaround += turnaround_time[i];
        total_waiting += waiting_time[i];
    }
    printf("Average Turnaround Time: %.2f\n", total_turnaround / n);
    printf("Average Waiting Time: %.2f\n", total_waiting / n);
    }

// SJF scheduling
void sjf(int n, int bt[], int at[]) {
    int completion_time[n];
    int turnaround_time[n];
    int waiting_time[n];
    bool completed[n];
    int current_time = 0;
    int completed_count = 0;

    for (int i = 0; i < n; completed[i] = false, i++) {}

    while (completed_count < n) {
        int min_burst = HIGH_PRIORITY;
        int selected = -1;
        for (int i = 0; i < n; i++) {
            if (!completed[i] && at[i] <= current_time && bt[i] < min_burst) {
            min_burst = bt[i];
```

```
            selected = i;
            }
        }

        if (selected == -1) {
            current_time++;
        }
        else {
            current_time += bt[selected];
            completion_time[selected] = current_time;
            turnaround_time[selected] = completion_time[selected] -
at[selected];
            waiting_time[selected] = turnaround_time[selected] - bt[selected];
            completed[selected] = true;
            completed_count++;
            printf("Process %d - Turnaround Time: %d, Waiting Time: %d\n",
selected + 1, turnaround_time[selected],waiting_time[selected]);
        }
    }

    float total_turnaround = 0.0f;
    float total_waiting = 0.0f;

    for (int i = 0; i < n; i++) {
        total_turnaround += turnaround_time[i];
        total_waiting += waiting_time[i];
    }
    printf("Average Turnaround Time: %.2f\n", total_turnaround / n);
    printf("Average Waiting Time: %.2f\n", total_waiting / n);
}

void priority_scheduling(int n, int bt[], int at[], int priority[]) {
    int completion_time[n];
    int turnaround_time[n];
    int waiting_time[n];
    bool completed[n];
    int current_time = 0;
    int completed_count = 0;

    for (int i = 0; i < n; completed[i] = false, i++) {}
        while (completed_count < n) {
            int min_priority = HIGH_PRIORITY;
            int selected = -1;
            for (int i = 0; i < n; i++) {
                if (!completed[i] && at[i] <= current_time && priority[i] <
min_priority) {
                    min_priority = priority[i];
                    selected = i;
```

```
                }
            }

            if (selected == -1) {
                current_time++;
            }
            else {
                current_time += bt[selected];
                completion_time[selected] = current_time;
                turnaround_time[selected] = completion_time[selected] -
at[selected];
                waiting_time[selected] = turnaround_time[selected] -
bt[selected];
                completed[selected] = true;
                completed_count++;
                printf("Process %d - Turnaround Time: %d, Waiting Time: %d\n",
selected + 1, turnaround_time[selected],waiting_time[selected]);
            }
        }

        float total_turnaround = 0.0f;
        float total_waiting = 0.0f;
        for (int i = 0; i < n; i++) {
            total_turnaround += turnaround_time[i];
            total_waiting += waiting_time[i];
        }
        printf("Average Turnaround Time: %.2f\n", total_turnaround / n);
        printf("Average Waiting Time: %.2f\n", total_waiting / n);
}

// Round Robin Scheduling
void round_robin(int n, int bt[], int at[], int time_quantum) {
    int remaining_time[n];
    int completion_time[n];
    int turnaround_time[n];
    int waiting_time[n];
    for (int i = 0; i < n; i++) {
        remaining_time[i] = bt[i];
    }

    int current_time = 0;
    bool finished = false;

    while (!finished) {
        finished = true;
        for (int i = 0; i < n; i++) {
            if (remaining_time[i] > 0 && current_time >= at[i]) {
                finished = false;
```

```
                int time_slice = (remaining_time[i] > time_quantum) ?
time_quantum : remaining_time[i];
                remaining_time[i] -= time_slice;
                current_time += time_slice;
                if (remaining_time[i] == 0) {
                    completion_time[i] = current_time;
                }
            }
        }
    }
    float total_turnaround = 0.0f;
    float total_waiting = 0.0f;

    for (int i = 0; i < n; i++) {
        turnaround_time[i] = completion_time[i] - at[i];
        waiting_time[i] = turnaround_time[i] - bt[i];
        total_turnaround += turnaround_time[i];
        total_waiting += waiting_time[i];
        printf("Process %d - Turnaround Time: %d, Waiting Time: %d\n", i + 1,
turnaround_time[i], waiting_time[i]);
    }

    float avg_turnaround = total_turnaround / n;
    float avg_waiting = total_waiting / n;
    printf("Average Turnaround Time: %.2f\n", avg_turnaround);
    printf("Average Waiting Time: %.2f\n", avg_waiting);
}
```