Cpu1.c

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

#define MAX_PROCESSES 10

typedef struct {
    int pid, arrival, burst, remaining, completion, waiting, turnaround;
} Process;

void calculate_avg_times(Process p[], int n) {
    float total_wait = 0, total_turnaround = 0;
    for (int i = 0; i < n; i++) {
        total_wait += p[i].waiting;
        total_turnaround += p[i].turnaround;
    }
    printf("\nAverage Waiting Time: %.2f", total_wait / n);
    printf("\nAverage Turnaround Time: %.2f\n", total_turnaround / n);
}

void fcfs(Process p[], int n) {
    int time = 0;
    for (int i = 0; i < n; i++) {
        if (time < p[i].arrival) time = p[i].arrival;
        time += p[i].burst;
        p[i].completion = time;
        p[i].turnaround = time - p[i].arrival;
        p[i].waiting = p[i].turnaround - p[i].burst;
    }
    printf("\nFCFS Scheduling:\n");
    calculate_avg_times(p, n);
}

void sjf_preemptive(Process p[], int n) {
    int completed = 0, time = 0;
    while (completed < n) {
        int shortest = -1, min_burst = INT_MAX;
        for (int i = 0; i < n; i++) {
            if (p[i].arrival <= time && p[i].remaining > 0 && p[i].remaining < min_burst) {
                min_burst = p[i].remaining;
                shortest = i;
            }
        }
        if (shortest == -1) { time++; continue; }

        p[shortest].remaining--;
        time++;
        if (p[shortest].remaining == 0) {
            p[shortest].completion = time;
            p[shortest].turnaround = time - p[shortest].arrival;
            p[shortest].waiting = p[shortest].turnaround - p[shortest].burst;
            completed++;
        }
    }
    printf("\nSJF (Preemptive) Scheduling:\n");
    calculate_avg_times(p, n);
}

void round_robin(Process p[], int n, int quantum) {
    int time = 0, remaining = n;
    while (remaining > 0) {
        for (int i = 0; i < n; i++) {
            if (p[i].remaining > 0 && p[i].arrival <= time) {
                int exec = (p[i].remaining > quantum) ? quantum : p[i].remaining;
                time += exec;
                p[i].remaining -= exec;
                if (p[i].remaining == 0) {
                    p[i].completion = time;
                    p[i].turnaround = time - p[i].arrival;
                    p[i].waiting = p[i].turnaround - p[i].burst;
                    remaining--;
                }
            }
        }
    }
    printf("\nRound Robin Scheduling:\n");
    calculate_avg_times(p, n);
}

int main() {
    int n, choice, quantum = 2;
    printf("Enter number of processes: ");
    scanf("%d", &n);

    Process p[n];
    printf("Enter Arrival and Burst Time for each process:\n");
    for (int i = 0; i < n; i++) {
        p[i].pid = i + 1;
        printf("Process %d\nArrival Time: ", p[i].pid);
        scanf("%d", &p[i].arrival);
        printf("Burst Time: ");
        scanf("%d", &p[i].burst);
        p[i].remaining = p[i].burst;
    }

    printf("\nSelect Scheduling Algorithm:\n1. FCFS\n2. SJF (Preemptive)\n3. Round Robin\
nChoice: ");
    scanf("%d", &choice);

    if (choice == 3) {
        printf("Enter Time Quantum: ");
        scanf("%d", &quantum);
    }

    switch (choice) {
        case 1: fcfs(p, n); break;
        case 2: sjf_preemptive(p, n); break;
        case 3: round_robin(p, n, quantum); break;
        default: printf("Invalid choice.\n"); return 1;
    }

    printf("\nProcess\tArrival\tBurst\tCompletion\tWaiting\tTurnaround\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t%d\t%d\t\t%d\t%d\n", p[i].pid, p[i].arrival, p[i].burst,
               p[i].completion, p[i].waiting, p[i].turnaround);
    }

    return 0;
}
```