Os lab

Multilevel queue

```
#include <stdio.h>
struct Process {
    int id, burst_time, arrival_time, queue;
    int waiting_time, turnaround_time, response_time;
};
void round_robin(struct Process p[], int n, int quantum) {
    int remaining_time[n], completed = 0, time = 0;
    for (int i = 0; i < n; i++) remaining_time[i] = p[i].burst_time;
    while (completed < n) {
        for (int i = 0; i < n; i++) {
            if (remaining_time[i] > 0) {
                if (remaining_time[i] > quantum) {
                    time += quantum;
                    remaining_time[i] -= quantum;
                } else {
                    time += remaining_time[i];
                    p[i].waiting_time = time - p[i].arrival_time - p[i].burst_time;
                    p[i].turnaround_time = time - p[i].arrival_time;
                    p[i].response_time = p[i].waiting_time;
                    remaining_time[i] = 0;
                    completed++;
                }
            }
        }
    }
}
void fcfs(struct Process p[], int n, int start_time) {
    int time = start_time;
    for (int i = 0; i < n; i++) {
        if (time < p[i].arrival_time)
            time = p[i].arrival_time;

        p[i].waiting_time = time - p[i].arrival_time;
        p[i].turnaround_time = p[i].waiting_time + p[i].burst_time;
        p[i].response_time = p[i].waiting_time;
        time += p[i].burst_time;
    }
```

```
}
int main() {
    int n;
    printf("Enter number of processes: ");
    scanf("%d", &n);

    struct Process processes[n], system_queue[n], user_queue[n];
    int sys_count = 0, user_count = 0;

    printf("Enter Burst Time, Arrival Time and Queue of each process: \n");
    for (int i = 0; i < n; i++) {
        printf("P%d: ", i + 1);
        scanf("%d %d %d", &processes[i].burst_time, &processes[i].arrival_time,
&processes[i].queue);
        processes[i]×id = i + 1;

        if (processes[i]×queue == 1)
            system_queue[sys_count++] = processes[i];
        else if (processes[i]×queue == 2)
            user_queue[user_count++] = processes[i];
    }
    int quantum = 2;
    round_robin(system_queue, sys_count, quantum);
    int last_exec_time = (sys_count > 0) ? system_queue[sys_count - 1].turnaround_time : 0;
    fcfs(user_queue, user_count, last_exec_time);

    printf("\nProcess\tWaiting Time\tTurn Around Time\tResponse Time\n");
    for (int i = 0; i < sys_count; i++)
        printf("P%d\t%d\t\t%d\t\t\t%d\n", system_queue[i].id, system_queue[i].waiting_time,
system_queue[i].turnaround_time, system_queue[i].response_time);

    for (int i = 0; i < user_count; i++)
        printf("P%d\t%d\t\t%d\t\t\t%d\n", user_queue[i].id, user_queue[i].waiting_time,
user_queue[i].turnaround_time, user_queue[i].response_time);

    float avg_wait = 0, avg_tat = 0, avg_resp = 0;
    for (int i = 0; i < sys_count; i++) {
        avg_wait += system_queue[i].waiting_time;
        avg_tat += system_queue[i].turnaround_time;
        avg_resp += system_queue[i].response_time;
    }
    for (int i = 0; i < user_count; i++) {
        avg_wait += user_queue[i].waiting_time;
        avg_tat += user_queue[i].turnaround_time;
        avg_resp += user_queue[i].response_time;
    }
    int total = sys_count + user_count;
    printf("\nAverage Waiting Time: %.2f", avg_wait / total);
    printf("\nAverage Turn Around Time: %.2f", avg_tat / total);
    printf("\nAverage Response Time: %.2f", avg_resp / total);
    printf("\nThroughput: %.2f\n", (float)total / avg_tat * total);
    return 0;
}
```

Rate monotonic

```
#include <stdio.h>

#define MAX_PROCESSES 10

typedef struct {
    int id;
    int burst_time;
    int period;
    int remaining_time;
    int next_deadline;
} Process;

void sort_by_period(Process processes[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (processes[j].period > processes[j + 1].period) {
                Process temp = processes[j];
                processes[j] = processes[j + 1];
                processes[j + 1] = temp;
            }
        }
    }
}

int gcd(int a, int b) {
    return b == 0 ? a : gcd(b, a % b);
}

int lcm(int a, int b) {
    return (a * b) / gcd(a, b);
}

int calculate_lcm(Process processes[], int n) {
    int result = processes[0]×period;
    for (int i = 1; i < n; i++) {
        result = lcm(result, processes[i].period);
    }
    return result;
}

double utilization_factor(Process processes[], int n) {
    double sum = 0;
    for (int i = 0; i < n; i++) {
        sum += (double)processes[i].burst_time / processes[i].period;
    }
    return sum;
}

double rms_threshold(int n) {
    return n * (pow(2.0, 1.0 / n) - 1);
}

void rate_monotonic_scheduling(Process processes[], int n) {
    int lcm_period = calculate_lcm(processes, n);
    printf("LCM=%d\n\n", lcm_period);

    printf("Rate Monotone Scheduling:\n");
    printf("PID  Burst  Period\n");
    for (int i = 0; i < n; i++) {
        printf("%d    %d      %d\n", processes[i].id, processes[i].burst_time, processes[i].period);
    }

    double utilization = utilization_factor(processes, n);
    double threshold = rms_threshold(n);
    printf("\n%.6f <= %.6f => %s\n", utilization, threshold, (utilization <= threshold) ? "true" :
"false");
```

```
    if (utilization > threshold) {
        printf("\nSystem may not be schedulable!\n");
        return;
    }

    int timeline = 0, executed = 0;
    while (timeline < lcm_period) {
        int selected = -1;
        for (int i = 0; i < n; i++) {
            if (timeline % processes[i]×period == 0) {
                processes[i].remaining_time = processes[i].burst_time;
            }
            if (processes[i].remaining_time > 0) {
                selected = i;
                break;
            }
        }
        if (selected != -1) {
            printf("Time %d: Process %d is running\n", timeline, processes[selected].id);
            processes[selected].remaining_time--;
            executed++;
        } else {
            printf("Time %d: CPU is idle\n", timeline);
        }
        timeline++;
    }
}

int main() {
    int n;
    Process processes[MAX_PROCESSES];

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

    printf("Enter the CPU burst times:\n");
    for (int i = 0; i < n; i++) {
        processes[i]×id = i + 1;
        scanf("%d", &processes[i].burst_time);
        processes[i].remaining_time = processes[i].burst_time;
    }

    printf("Enter the time periods:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &processes[i].period);
    }

    sort_by_period(processes, n);
    rate_monotonic_scheduling(processes, n);

    return 0;
}
```

Earliest

```
#include <stdio.h>

int gcd(int a, int b) {
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}

int lcm(int a, int b) {
    return (a * b) / gcd(a, b);
}

struct Process {
    int id, burst_time, deadline, period;
};

void earliest_deadline_first(struct Process p[], int n, int time_limit) {
    int time = 0;
    printf("Earliest Deadline Scheduling:\n");
    printf("PID\tBurst\tDeadline\tPeriod\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\n", p[i].id, p[i].burst_time, p[i].deadline, p[i].period);
    }
    printf("\nScheduling occurs for %d ms\n", time_limit);
    while (time < time_limit) {
        int earliest = -1;
        for (int i = 0; i < n; i++) {
            if (p[i].burst_time > 0) {
                if (earliest == -1 || p[i].deadline < p[earliest].deadline) {
                    earliest = i;
                }
            }
        }

        if (earliest == -1) break;

        printf("%dms: Task %d is running.\n", time, p[earliest].id);
        p[earliest].burst_time--;
        time++;
    }
}

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

    struct Process processes[n];
    printf("Enter the CPU burst times:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &processes[i].burst_time);
        processes[i]×id = i + 1;
    }

    printf("Enter the deadlines:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &processes[i].deadline);
    }

    printf("Enter the time periods:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &processes[i].period);
    }
    int hyperperiod = processes[0]×period;
    for (int i = 1; i < n; i++) {
        hyperperiod = lcm(hyperperiod, processes[i].period);
    }

    printf("\nSystem will execute for hyperperiod (LCM of periods): %d ms\n", hyperperiod);

    earliest_deadline_first(processes, n, hyperperiod);

    return 0;
}
```