Lab 3

1.

#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\tWT\tTAT\tRt\n");

for (int i = 0; i < sys\_count; i++)

printf("P%d\t%d\t%d\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%d\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);

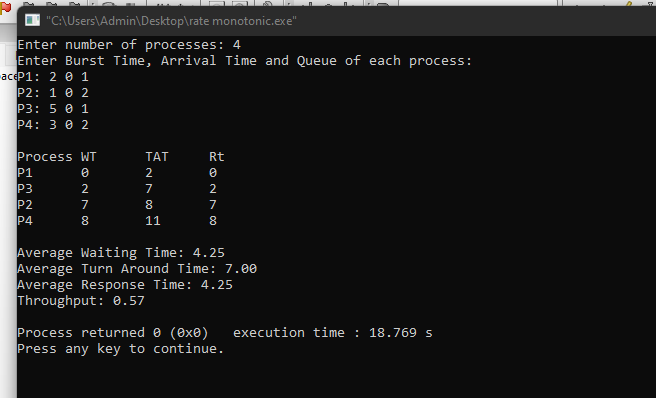
printf("\nAverage Response Time: %.2f", avg\_resp / total);

printf("\nThroughput: %.2f\n", (float)total / avg\_tat \* total);

return 0;

}

Output



Lab 4

1.

#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);

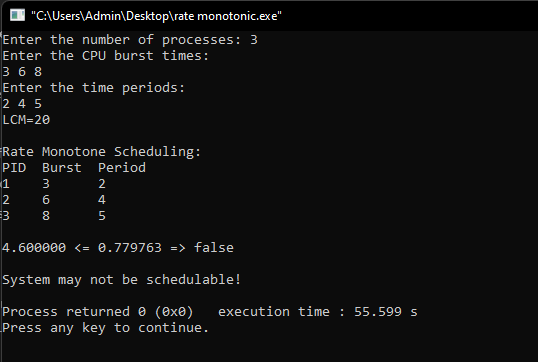
}

sort\_by\_period(processes, n);

rate\_monotonic\_scheduling(processes, n);

return 0;

}



2.

#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;  
}

Output:

