Week 3

#include <stdio.h>

#include <math.h>

#define MAX\_TASKS 10

typedef struct {

int id;

int period;

int exec\_time;

int remaining\_time;

int deadline;

} Task;

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 compute\_hyperperiod(Task tasks[], int n) {

int hyperperiod = tasks[0].period;

for (int i = 1; i < n; i++) {

hyperperiod = lcm(hyperperiod, tasks[i].period);

}

return hyperperiod;

}

void sort\_tasks(Task tasks[], int n) {

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (tasks[j].period > tasks[j + 1].period) {

Task temp = tasks[j];

tasks[j] = tasks[j + 1];

tasks[j + 1] = temp;

}

}

}

}

int is\_schedulable(Task tasks[], int n) {

double utilization = 0.0, bound;

for (int i = 0; i < n; i++) {

utilization += (double)tasks[i].exec\_time / tasks[i].period;

}

bound = n \* (pow(2.0, 1.0 / n) - 1);

printf("\nCPU Utilization = %.4lf", utilization);

printf("\nUtilization Bound for %d tasks = %.4lf", n, bound);

if (utilization <= bound) {

printf("\nTask set can be scheduled under RMS.\n");

return 1;

} else {

printf("\nTask set cannot be scheduled under RMS.\n");

return 0;

}

}

void rate\_monotonic(Task tasks[], int n) {

sort\_tasks(tasks, n);

int hyperperiod = compute\_hyperperiod(tasks, n);

printf("\nRate-Monotonic Scheduling = %d\n", hyperperiod);

for (int t = 0; t < hyperperiod; t++) {

int current\_task = -1;

for (int i = 0; i < n; i++) {

if (t % tasks[i].period == 0) {

tasks[i].remaining\_time = tasks[i].exec\_time;

}

if (tasks[i].remaining\_time > 0) {

if (current\_task == -1 || tasks[i].period < tasks[current\_task].period) {

current\_task = i;

}

}

}

if (current\_task != -1) {

printf("Time %d: Executing Task %d\n", t, tasks[current\_task].id);

tasks[current\_task].remaining\_time--;

} else {

printf("Time %d: CPU Idle\n", t);

}

}

}

int main() {

int n;

printf("Enter the number of processes: ");

scanf("%d", &n);

Task tasks[MAX\_TASKS];

for (int i = 0; i < n; i++) {

printf("\nEnter details for Process %d:\n", i + 1);

tasks[i].id = i + 1;

printf("Period: ");

scanf("%d", &tasks[i].period);

printf("Execution Time: ");

scanf("%d", &tasks[i].exec\_time);

tasks[i].remaining\_time = 0;

}

if (is\_schedulable(tasks, n)) {

rate\_monotonic(tasks, n);

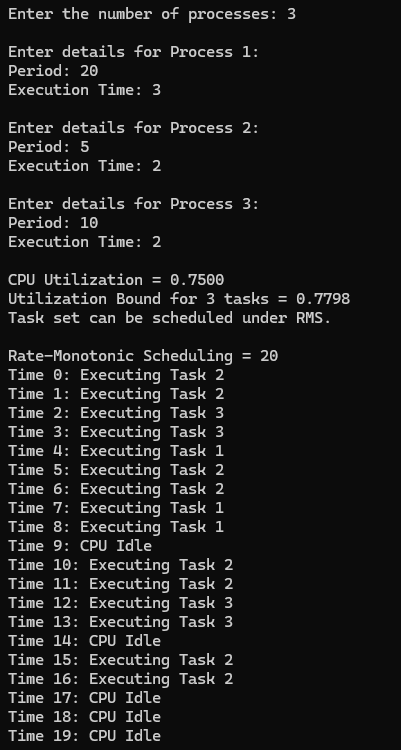
} else {

printf("\n Utilization is above the threshold!\n");

}

return 0;

}



#include <stdio.h>

#include <stdlib.h>

#define MAX\_TASKS 10

typedef struct {

int id;

int period;

int exec\_time;

int remaining\_time;

int next\_deadline;

} Task;

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 compute\_hyperperiod(Task tasks[], int n) {

int hyperperiod = tasks[0].period;

for (int i = 1; i < n; i++) {

hyperperiod = lcm(hyperperiod, tasks[i].period);

}

return hyperperiod;

}

void earliest\_deadline\_first(Task tasks[], int n) {

int hyperperiod = compute\_hyperperiod(tasks, n);

printf("\nSimulating Earliest-Deadline First (EDF) Scheduling for Hyperperiod = %d\n", hyperperiod);

for (int i = 0; i < n; i++) {

tasks[i].next\_deadline = tasks[i].period;

tasks[i].remaining\_time = 0;

}

for (int t = 0; t < hyperperiod; t++) {

int current\_task = -1;

int min\_deadline = hyperperiod + 1;

for (int i = 0; i < n; i++) {

if (t % tasks[i].period == 0) {

tasks[i].remaining\_time = tasks[i].exec\_time;

tasks[i].next\_deadline = t + tasks[i].period;

}

if (tasks[i].remaining\_time > 0 && tasks[i].next\_deadline < min\_deadline) {

current\_task = i;

min\_deadline = tasks[i].next\_deadline;

}

}

if (current\_task != -1) {

printf("Time %d: Executing Task %d (Deadline at %d)\n", t, tasks[current\_task].id, tasks[current\_task].next\_deadline);

tasks[current\_task].remaining\_time--;

} else {

printf("Time %d: CPU Idle\n", t);

}

}

}

int main() {

int n;

printf("Enter the number of tasks: ");

scanf("%d", &n);

Task tasks[MAX\_TASKS];

for (int i = 0; i < n; i++) {

printf("\nEnter details for Task %d:\n", i + 1);

tasks[i].id = i + 1;

printf("Period: ");

scanf("%d", &tasks[i].period);

printf("Deadline: ");

scanf("%d", &tasks[i].next\_deadline);

printf("Execution Time: ");

scanf("%d", &tasks[i].exec\_time);

}

earliest\_deadline\_first(tasks, n);

return 0;

}

