Dining philosopher problem

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

#define N 5

enum { THINKING, HUNGRY, EATING };

int state[N];

sem\_t mutex;

sem\_t sem[N];

void test(int i) {

int left = (i + N - 1) % N;

int right = (i + 1) % N;

if (state[i] == HUNGRY &&

state[left] != EATING &&

state[right] != EATING) {

state[i] = EATING;

sem\_post(&sem[i]);

}

}

void take\_forks(int i) {

sem\_wait(&mutex);

state[i] = HUNGRY;

printf("Philosopher %d is HUNGRY\n", i);

test(i);

sem\_post(&mutex);

sem\_wait(&sem[i]);

}

void put\_forks(int i) {

sem\_wait(&mutex);

state[i] = THINKING;

printf("Philosopher %d is THINKING\n", i);

test((i + N - 1) % N);

test((i + 1) % N);

sem\_post(&mutex);

}

void\* philosopher(void\* num) {

int id = \*(int\*)num;

while (1) {

sleep(rand() % 3 + 1);

take\_forks(id);

printf("Philosopher %d is EATING\n", id);

sleep(rand() % 2 + 1);

put\_forks(id);

}

}

int main() {

pthread\_t tid[N];

int ids[N];

sem\_init(&mutex, 0, 1);

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

sem\_init(&sem[i], 0, 0);

state[i] = THINKING;

ids[i] = i;

}

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

pthread\_create(&tid[i], NULL, philosopher, &ids[i]);

}

sleep(20);

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

pthread\_cancel(tid[i]);

sem\_destroy(&sem[i]);

}

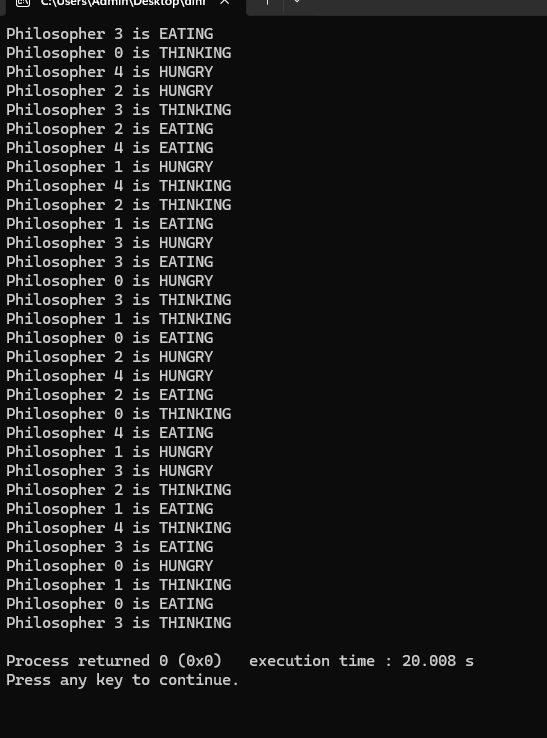
sem\_destroy(&mutex);

return 0;

}

Output:





Producer-Consumer problem using semaphores.

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

#define BUFFER\_SIZE 5

#define PRODUCE\_COUNT 10

int buffer[BUFFER\_SIZE];

int in = 0, out = 0;

sem\_t empty, full, mutex;

void\* producer(void\* arg) {

for (int i = 0; i < PRODUCE\_COUNT; i++) {

int item = rand() % 100;

sem\_wait(&empty);

sem\_wait(&mutex);

buffer[in] = item;

printf("Produced: %d at buffer[%d]\n", item, in);

in = (in + 1) % BUFFER\_SIZE;

sem\_post(&mutex);

sem\_post(&full);

sleep(1); // simulate work

}

pthread\_exit(NULL);

}

void\* consumer(void\* arg) {

for (int i = 0; i < PRODUCE\_COUNT; i++) {

sem\_wait(&full);

sem\_wait(&mutex);

int item = buffer[out];

printf("Consumed: %d from buffer[%d]\n", item, out);

out = (out + 1) % BUFFER\_SIZE;

sem\_post(&mutex);

sem\_post(&empty);

sleep(2);

}

pthread\_exit(NULL);

}

int main() {

pthread\_t prod\_thread, cons\_thread;

sem\_init(&empty, 0, BUFFER\_SIZE);

sem\_init(&full, 0, 0);

sem\_init(&mutex, 0, 1);

pthread\_create(&prod\_thread, NULL, producer, NULL);

pthread\_create(&cons\_thread, NULL, consumer, NULL);

pthread\_join(prod\_thread, NULL);

pthread\_join(cons\_thread, NULL);

sem\_destroy(&empty);

sem\_destroy(&full);

sem\_destroy(&mutex);

return 0;

}



Earliest-deadline First

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

#define MAX\_TIME 100

typedef struct {

int id;

int period;

int execution\_time;

int remaining\_time;

int next\_release\_time;

int absolute\_deadline;

} Task;

int simulate\_edf(Task tasks[], int n, int time\_limit) {

char timeline[time\_limit];

for (int t = 0; t < time\_limit; t++) timeline[t] = ' ';

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

tasks[i].next\_release\_time = 0;

tasks[i].remaining\_time = 0;

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

}

for (int time = 0; time < time\_limit; time++) {

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

if (time == tasks[i].next\_release\_time) {

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

printf("Task %d missed its deadline at time %d!\n", tasks[i].id, time);

return 0;

}

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

tasks[i].absolute\_deadline = time + tasks[i].period;

tasks[i].next\_release\_time += tasks[i].period;

}

}

int selected = -1;

int earliest\_deadline = INT\_MAX;

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

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

earliest\_deadline = tasks[i].absolute\_deadline;

selected = i;

}

}

if (selected != -1) {

tasks[selected].remaining\_time--;

timeline[time] = '0' + tasks[selected].id;

} else {

timeline[time] = '-';

}

}

printf("\nTimeline:\n");

for (int t = 0; t < time\_limit; t++) {

if (t % 10 == 0) printf("%3d", t);

else printf(" ");

}

printf("\n");

for (int t = 0; t < time\_limit; t++) {

printf(" %c", timeline[t]);

}

printf("\n");

return 1;

}

int main() {

int n;

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

scanf("%d", &n);

Task tasks[n];

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

printf("Enter execution time and period for Task %d: ", i + 1);

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

tasks[i].id = i + 1;

tasks[i].remaining\_time = 0;

}

int time\_limit = MAX\_TIME;

int result = simulate\_edf(tasks, n, time\_limit);

if (result == 1) {

printf("\nAll tasks completed within their deadlines!\n");

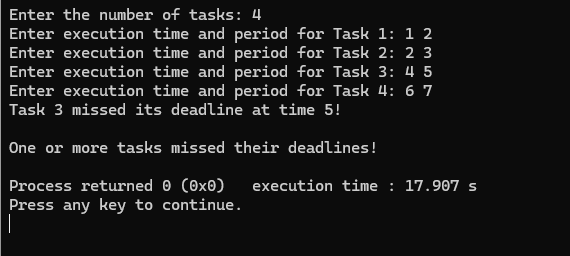
} else {

printf("\nOne or more tasks missed their deadlines!\n");

}

return 0;

}



Rate- Monotonic

#include <stdio.h>

#include <stdlib.h>

#define MAX\_TIME 100

typedef struct {

int id;

int period;

int execution\_time;

int remaining\_time;

int next\_release\_time;

int last\_run\_time;

} Task;

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 simulate\_rms(Task tasks[], int n, int time\_limit) {

char timeline[time\_limit];

for (int t = 0; t < time\_limit; t++) timeline[t] = ' ';

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

tasks[i].next\_release\_time = 0;

tasks[i].remaining\_time = 0;

}

for (int time = 0; time < time\_limit; time++) {

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

if (time == tasks[i].next\_release\_time) {

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

printf("Task %d missed its deadline at time %d!\n", tasks[i].id, time);

return 0;

}

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

tasks[i].next\_release\_time += tasks[i].period;

}

}

int selected = -1;

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

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

selected = i;

break;

}

}

if (selected != -1) {

tasks[selected].remaining\_time--;

timeline[time] = '0' + tasks[selected].id;

} else {

timeline[time] = '-';

}

}

printf("\nTimeline:\n");

for (int t = 0; t < time\_limit; t++) {

if (t % 10 == 0) printf("%3d", t);

else printf(" ");

}

printf("\n");

for (int t = 0; t < time\_limit; t++) {

printf(" %c", timeline[t]);

}

printf("\n");

return 1;

}

int main() {

int n;

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

scanf("%d", &n);

Task tasks[n];

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

printf("Enter execution time and period for Task %d: ", i + 1);

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

tasks[i].id = i + 1;

tasks[i].remaining\_time = 0;

tasks[i].last\_run\_time = -1;

}

sort\_tasks(tasks, n);

int time\_limit = MAX\_TIME;

int result = simulate\_rms(tasks, n, 50);

if (result == 1) {

printf("\nAll tasks completed within their deadlines!\n");

} else {

printf("\nOne or more tasks missed their deadlines!\n");

}

return 0;

}

