**מעבדה 10 -** קיפאון (Deadlock)

**מגישים:** 1. חוטמליאנסקי דמיטרי 334017415

2. שיח אחמד מוחמד 209158120

תרגיל 1

#include <pthread.h>

#include <stdio.h>

#include <string.h>

#include <semaphore.h>

#include <time.h>

#include <stdlib.h>

#define N 4 //amount of cities

sem\_t cities[N]; //semaphores for each city

int haifa\_zfat\_talks = 0; //flag

void\* Func(void \* c\_num); //function generate random number 0-3

// and calls to Calls function

void Calls(int City1, int City2);

int main() {

pthread\_t thread[N]; //threads array

int c\_num[N]; //save the indexes of the cities

int i, ans;

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

sem\_init(&cities[i], 0, 1); //cities semaphores initialization

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

c\_num[i] = i;

ans = pthread\_create(&thread[i], NULL, Func, (void\*)&c\_num[i]);

if (ans != 0){ //check if thread created successful

printf("failed to create a thread");

exit(1);

}

}

sleep(20);

return 0;

}

void\* Func(void \*c\_num)

{

int c1 = \*((int\*)c\_num); //c1 = City1

int c2; //c2 = City2

while (1) {

c2 = rand() % N;

if (c1 != c2) //if City1 and City2 are not the same city

Calls(c1, c2); //call to CALLS function

}

}

void Calls(int City1, int City2) {

int higher, lower; //for city indexes

int i;

if (City1 < City2) {

lower = City1;

higher = City2;

}

else {

lower = City2;

higher = City1;

}

if ((lower == 2 && higher == 3)) { //if cities are Haifa or Zfat

haifa\_zfat\_talks = 1; //set the flag 1

for(i = 0; i < N; i++) //all the cities waits the end of talking

sem\_wait(&cities[i]); //of Haifa with Zfat

printf("%d started to talk with %d\n", City1, City2);

sleep(1);

printf("%d finished to talk with %d\n", City1, City2);

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

sem\_post(&cities[i]); //all the cities continue their calling

haifa\_zfat\_talks = 0; //set the flag 0

}

else {

if(!haifa\_zfat\_talks){ //if the flag = 0 another cities may calling

sem\_wait(&cities[lower]);

sem\_wait(&cities[higher]);

printf("%d started to talk with %d\n", City1, City2);

sleep(1);

printf("%d finished to talk with %d\n", City1, City2);

sem\_post(&cities[lower]);

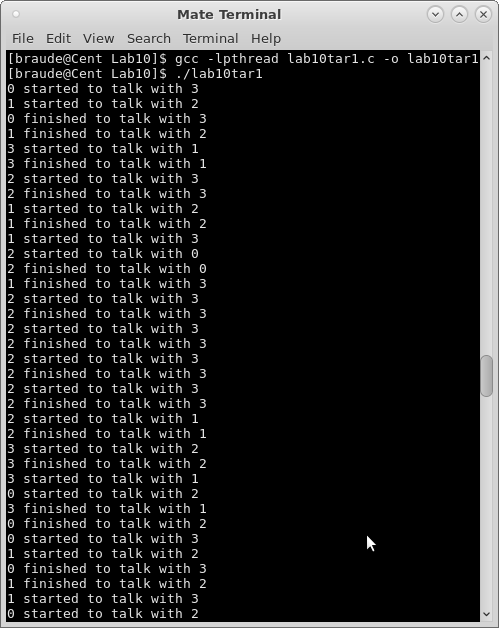
sem\_post(&cities[higher]);

//end of calling of two cities

}

}

}



תרגיל 2

#include <stdio.h>

#include <pthread.h>

#include <unistd.h>

#include <assert.h>

#include <time.h>

#include <stdlib.h>

#include <semaphore.h>

#include <string.h>

#define LEFT (i + N - 1) % N //מקרו המגדיר את מספר השכן משמאל

#define RIGHT (i + 1) % N //מקרו המגדיר את מספר השכן מימין

#define THINKING 0 //פילוסוף חושב

#define HUNGRY 1 //פילוסוף רעב (בהמתנה למקלות

#define EATING 2 //פילוסוף אוכל

#define MAX\_TIME 5 //זמן מקסימלי שמותר לבזבז על אוכל ו/או חשיבה

int N; //מספר פילוסופים

sem\_t \*mutex = NULL; // מצביע לסמפור בינארי

sem\_t \*eaters = NULL; //מערך סמפורים של פילוסופים

int\* state = NULL; //מערך שלמים לקביעת מצב פילוסוף (אוכל, רעב, חושב

/\*\*\*\*\*\*\*\*\*\* eating function \*\*\*\*\*\*\*\*\*/

void eat(int i) {

int time = rand() % MAX\_TIME;

printf("Philosopher #%d is eating...\n", i + 1);

sleep(time);

printf("Philosopher #%d stopped eating...\n", i + 1);

}

/\*\*\*\*\*\*\*\*\*\* thinking function \*\*\*\*\*\*\*\*\*/

void think(int i) {

int time = rand() % MAX\_TIME;

printf("Philosopher #%d is thinking...\n", i + 1);

sleep(time);

}

/\*\*\*\*\*\*\*\*\*\* test if HUNGRY may eating? \*\*\*\*\*\*\*\*/

void test(int i) {

/\*\*\*\* checks if hungry and sticks free \*\*\*\*/

if (state[i] == HUNGRY && state[LEFT] != EATING && state[RIGHT] != EATING)

{

state[i] = EATING;

sem\_post(&eaters[i]);

}

}

/\*\*\*\*\*\*\*\*\* try to take the sticks \*\*\*\*\*\*\*\*\*/

void take\_sticks(int i) {

sem\_wait(mutex); //נכנסים לקטע קריטי ונועלים אותו כדי שהמקלות יוכל לקחת רק פילוסוף אחד

state[i]=HUNGRY; //הפילוסוף רעב

test(i); //מנסה לקחת 2 מקלות בעזרת פונקציית טסט

sem\_post(mutex); //עוזבים את הקטע הקריטי

sem\_wait(&eaters[i]); // אם לא הצליח להתחיל לאכול, ננעל. השכנים יוכלו "לפתוח" אותו אחרי שיוכלו

}

/\*\*\*\*\*\*\*\* try to return the stick \*\*\*\*\*\*\*\*\*/

void put\_sticks(int i) {

sem\_wait(mutex); //נכנסים לקטע קריטי ונועלים אותו כדי שרק פילוסוף אחד יעבוד עם המקלות

state[i]=THINKING; //פילוסוף סיים לאכול, התחיל לחשוב

test(LEFT); //בודקים האם השכן משמאל רעב, ז"א, ממתין למקל, אז נותנים לו סימן שיתחיל לאכול

test(RIGHT); //כנ"ל עם השכן מימין

sem\_post(mutex); // בשביל מה?

}

/\* a philosoph is a thread \*/

void\* philosopher(void\* arg) {

int i = \*((int\*)arg);

while (1) { //לולאה אין סופית

think(i); //קודם כל הפילוסוף חושב

take\_sticks(i); //אחר כך מנסה לקחת את המקלות

eat(i); //אוכל

put\_sticks(i); //שם את המקלות על השולחן

}

}

int main(int argc, char\* argv[]) {

void\* result;

if (argc != 2) {

fprintf(stderr, "Usage: %s <NUMBER\_OF\_PHILOSOPHERS>\n", argv[0]);

return 1;

}

N = atoi(argv[1]); // המרת כמות הפילוסופים ממחרוזת למספר שלם

if (N <= 1){

fprintf(stderr,"Error by transformation of the argument...\n");

return 2; //לא פחות מ-2 פילוסופים

}

mutex = (sem\_t\*)malloc(sizeof(sem\_t));

if (sem\_init(mutex, 0, 1) != 0) { //יוצרים מיוטקס להפרדה בכניסה לקטע קריטי

fprintf(stderr, "Error by creating semaphore...\n");

return 3;

}

eaters = (sem\_t\*)malloc(N\*sizeof(sem\_t)); //מקצים זיכרון לכמות סמפורים - לכל פילוסוף סמפור

state = (int\*)malloc(N\*sizeof(int)); //מקצים זיכרון למערך שלמים, כל שלם מייצג מצב פילוסוף מתאים (רעב, אוכל,חושב

memset(state, 0, N);

srand(time(NULL));

pthread\_t \*philosophers = (pthread\_t\*)malloc(N \* sizeof(pthread\_t)); //מקצים זיכרון לחוטים לפי כמות הפילוסופים

int i;

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

if (sem\_init(&eaters[i], 0, 0) != 0) { //מאתחלים סמפורים של פילוסופים

fprintf(stderr, "Error by creating semaphore...\n"); return 3;

}

}

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

if (pthread\_create(&philosophers[i], NULL, philosopher, (void\*)&i) != 0) {

fprintf(stderr, "Error by creating thread\n");

return 2;

}

usleep(100000);

}

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

if (pthread\_join(philosophers[i], &result) != 0) {

fprintf(stderr, "Error by joining thread\n");

return 3;

}

return 0;

}

}

