Міністерство освіти і науки України

Львівський національний університет імені Івана Франка

Факультет електроніки та комп’ютерних технологій

Кафедра радіоелектронних і комп’ютерних систем

Звіт

про виконання лабораторних робіт №7

“Програмна реалізація міжпотокової взаємодії в ОС Windows і Linux”

Виконав

студент групи ФеІ-23

Гупало Мар’ян

Перевірив

ac. Сінькевич О. О.

Львів – 2019

**Завдання №1.**

Напишіть код таких функцій:

а) send\_msg(), що відсилає повідомлення i N потокам і припиняє поточний потік, поки усі вони не одержать повідомлення;

б) recv\_msg(), що припиняє даний потік до одержання відісланого за допомогою send\_msg () повідомлення.

Використовуйте потоки POSIX і Win32. Для потоків Win32 моделюйте умовні змінні з використанням подій.

Thread.c

#include <stdio.h>

#include <pthread.h>

pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

pthread\_cond\_t cond = PTHREAD\_COND\_INITIALIZER;

int condition = 0;

void send\_msg()

{

 condition = 1;

 pthread\_cond\_broadcast(&cond);

}

void resive\_msg()

{

 pthread\_mutex\_lock(&mutex);

 while(!condition)

 pthread\_cond\_wait(&cond, &mutex);

 pthread\_mutex\_unlock(&mutex);

}

void\* someThread(void\* p)

{

 int threadID = \*(int\*) p;

 printf("Thread %i is waiting for message ...\n", threadID);

 resive\_msg();

 printf("Thread %i reseive message ...\n", threadID);

}

    int main()

{

 int THREAD\_CONUT = 5;

 pthread\_t threads[THREAD\_CONUT];

 for(int i = 0; i < THREAD\_CONUT; ++i)

 {

 printf("Thread %i is starting\n", i);

 pthread\_create(&threads[i], NULL, someThread, &i);

 }

 printf("Message sent for all threads\n");

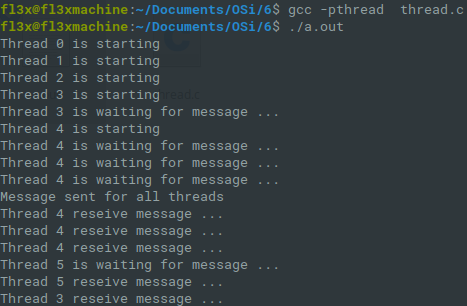
 send\_msg();

 for(int i = 0; i < THREAD\_CONUT; ++i)

pthread\_join(threads[i], NULL);

 return 0;

}



**Завдання №2**

Реалізуйте спільно використовувану динамічну структуру даних (стек, двозв'язний список, бінарне дерево) з використанням потоків POSIX і Win32. Функції доступу до цієї структури даних оформіть, якщо це можливо, у вигляді монітора.

#include <stdio.h>

#include <pthread.h>

#include <math.h>

#include <stdlib.h>

pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

int\* stack;

int size;

int currentPos = 0;

void createStack(int s){

size = s;

stack = malloc(size\*sizeof(\*stack));

}

int push(int value) {

pthread\_mutex\_lock(&mutex);

if (currentPos >= size) {

pthread\_mutex\_unlock(&mutex);

return -1;

}

stack[currentPos++] = value;

pthread\_mutex\_unlock(&mutex);

}

int pop() {

pthread\_mutex\_lock(&mutex);

if (currentPos <= 0) {

pthread\_mutex\_unlock(&mutex);

return -1;

}

currentPos--;

pthread\_mutex\_unlock(&mutex);

return stack[currentPos];

}

int ind = 0;

pthread\_mutex\_t i = PTHREAD\_MUTEX\_INITIALIZER;

int getIndex(){

ind++;

return ind;

}

void\* threadFunc(void\* p) {

pthread\_mutex\_lock(&i);

int value = getIndex();

pthread\_mutex\_unlock(&i);

int currentVal = value;

for (int i = 1; i < 6; ++i){

if (push(currentVal) != -1)

printf("Pushed %i \n", currentVal);

else

printf("Stack is full!\n");

currentVal = value \*pow(5,i) + currentVal;

}

for (int i = 0; i < 6; ++i)

if ((currentVal = pop()) != -1)

printf("Pop %i \n", currentVal);

else

printf("Stack is empty!\n");

}

int main() {

int THREAD\_COUNT = 3;

pthread\_t threads[THREAD\_COUNT];

createStack(THREAD\_COUNT \* 5);

for (int i = 1; i <= THREAD\_COUNT; ++i) {

pthread\_create(&threads[i - 1], NULL, threadFunc, NULL);

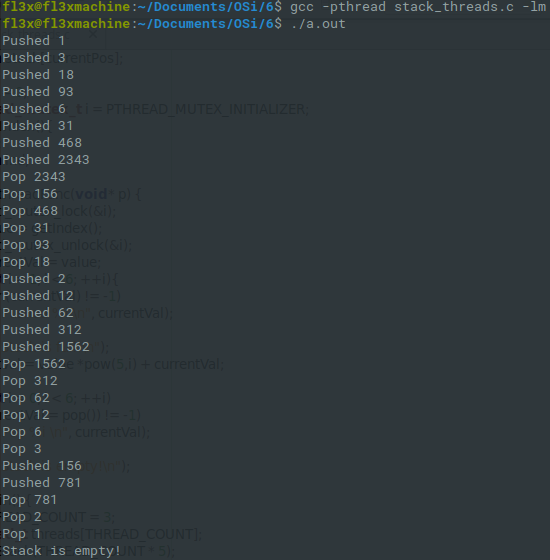
}

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

pthread\_join(threads[i], NULL);

return 0;

}



Висновок: на даній лабораторній роботі я навився користуватись потоками, блокувати використання функцій багатьма потоками мутексами, реалізував стек на основі багатопоточності.