Міністерство освіти і науки України
Львівський національний університет імені Івана Франка

*Факультет електроніки*
*Кафедра радіоелектроніки та комп'ютерних технологій*

# Звіт
# про виконання лабораторної роботи №6
# на тему: «Програмна реалізація міжпотокової взаємодії в ОС Windows і Linux»

Виконав студент
групи ФЕІ - 23
Попович Василь
Викладач:
Сінькевич О. О.

Львів – 2019

**Мета:** ознайомитись з механізмами міжпотокової взаємодії.

**Завдання до виконання:**

1. Напишіть код таких функцій:

а) send_msg(), що відсилає повідомлення і 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.

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.

3. Розробіть програму реалізації блокувань читання-записування з перевагою записування. Використайте потоки POSIX.

**Теоретична частина:**

Потоки, які виконуються в рамках процесу паралельно, можуть бути незалежними або взаємодіяти між собою. Потік є незалежним, якщо він не впливає на виконання інших потоків процесу, не зазнає впливу з їхнього боку, та не має з ними жодних спільних даних. Його виконання однозначно залежить від вхідних даних і називається детермінованим. Усі інші потоки є такими, що взаємодіють. Ці потоки мають дані, спільні з іншими потоками (вони перебувають в адресному просторі їхнього процесу), їх виконання залежить не тільки від вхідних даних, але й від виконання інших потоків, тобто вони є

недетермінованими. Дані, які є загальними для кількох потоків, називають спільно використовуваними даними (shared data). Механізми забезпечення коректною доступу до спільно використовуваних даних називають механізмами синхронізації потоків.Основні проблеми взаємодії потоків Проблема змагання

Критичні секції та блокування розв'язанням проблеми змагання є перетворення фрагмента коду, який спричиняє проблему, в атомарну операцію, тобто в таку, котра гарантовано виконуватиметься цілковито без втручання інших потоків. Такий фрагмент коду називають критичною секцією (critical section):

```
// початок критичної секції

total_amount = total_amount + new_amount;
```

// кінець критичної секції

Розглянемо властивості, які повинна мати критична секція.

♦ Взаємного виключення (mutual exclusion): у конкретний момент часу код критичної секції може виконувати тільки один потік.

♦ Прогресу: якщо кілька потоків одночасно запросили вхід у критичну секцію, один із них повинен обов'язково у неї ввійти (вони не можуть всі заблокувати один одного).

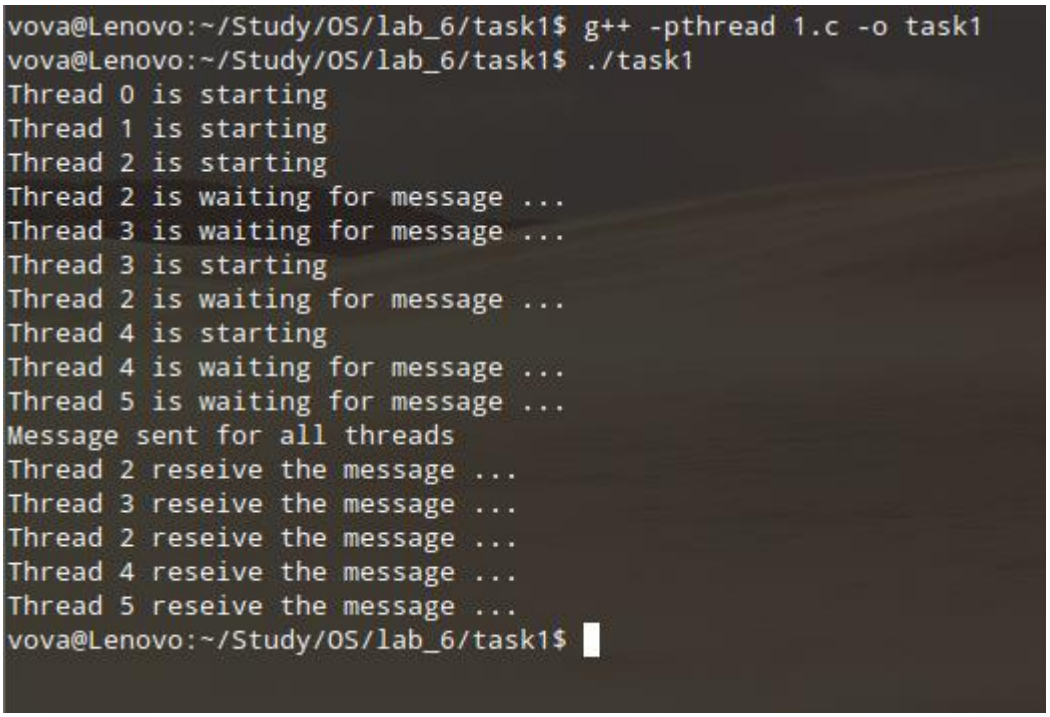
♦ Обмеженості очікування: процес, що намагається ввійти у критичну секцію, рано чи пізно обов'язково в неї ввійде.

**Виконання:**

**Завдання 1:** В цій програмі створюється п’ять додаткових потоків, в тілі яких викликається resive_msg. Після цього основний потік викликає send_msg. Тобто всі дочірні потоки будуть очікавати, щоб продовжити виконання, поки не буде відправлено повідомлення.

```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 the 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");
```

Приклад роботи програми:




На цьому рисунку ми бачимо, що програма виводить нам повідомлення про стан потоків. В кінцевому результаті всі 5 потоків отримають повідомлення.

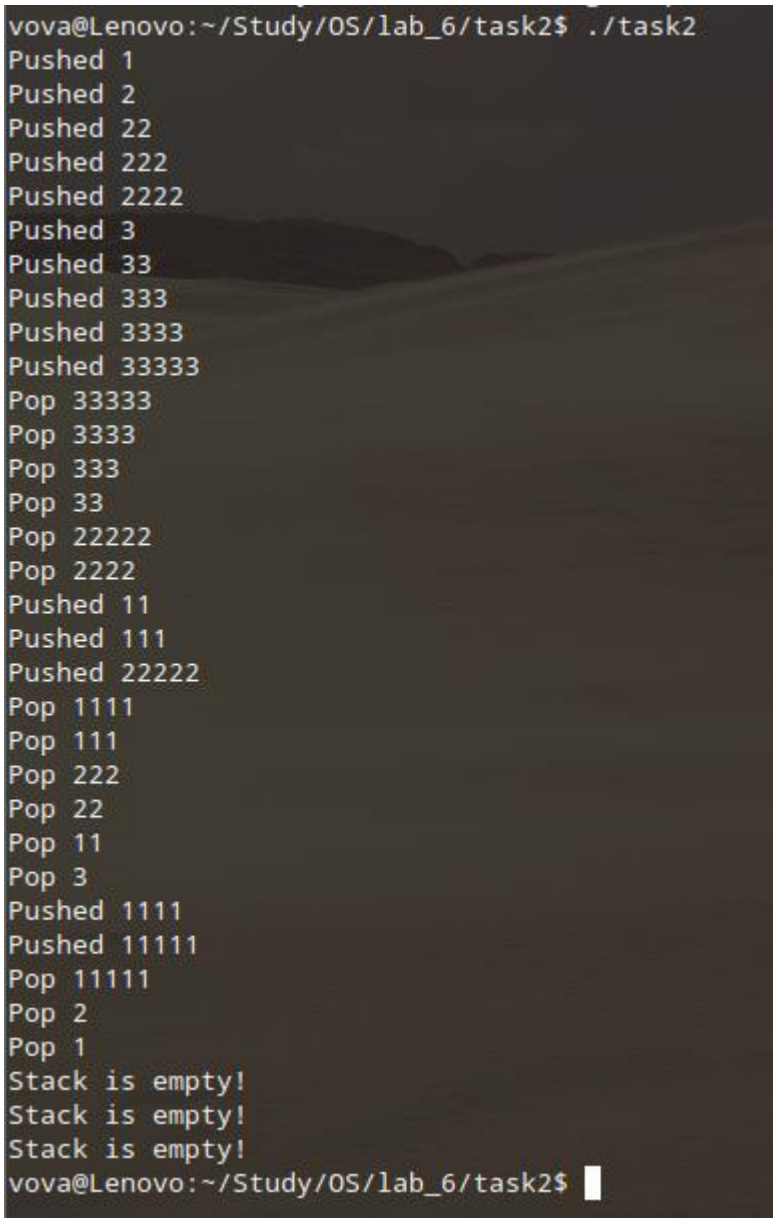
**Завдання 2:**

У цій частині роботи потрібно було реалізувати спільно реалізовано динамічну структуру даних. Я реалізував це на прикладі стеку. У стеку є дві основні функції – push (додати елемент) і pop (видалити елемент).

У цій програмі створені три потоки, які спільно використовують стек і мають індекси 1-3.Числа, які вони записують в стек позначають кількома цифрами. Наприклад, 33 означає, що ці два перші числа записані третім потоком. А число 2 означає, що перше число записане другим потоком.

```
#include <stdio.h>
#include <pthread.h>
#include <math.h>

//Synchronized stack implementation
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
int* stack;
int size;
int currentPos = 0;

void createStack(int s){
    size = s;
    stack = new int[size];
}

int push(int value) { // -1 if value doesn`t pushed (stack is full)
    pthread_mutex_lock(&mutex);
    if (currentPos >= size) {
        pthread_mutex_unlock(&mutex);
        return -1;
    }
    stack[currentPos++] = value;
    pthread_mutex_unlock(&mutex);
}

int pop() { // -1 if stack is empty
    pthread_mutex_lock(&mutex);
    if (currentPos <= 0) {
        pthread_mutex_unlock(&mutex);
        return -1;
    }
    currentPos--;
    pthread_mutex_unlock(&mutex);
    return stack[currentPos];

}
int index = 0;
pthread_mutex_t i = PTHREAD_MUTEX_INITIALIZER;
int getIndex(){
    index++;
    return index;
}
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(10,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);
```

Приклад роботи програми:




З рисунку видно, що у стек спочатку додавалися елементи, а потім видалялися, в результаті чого стек став порожнім.

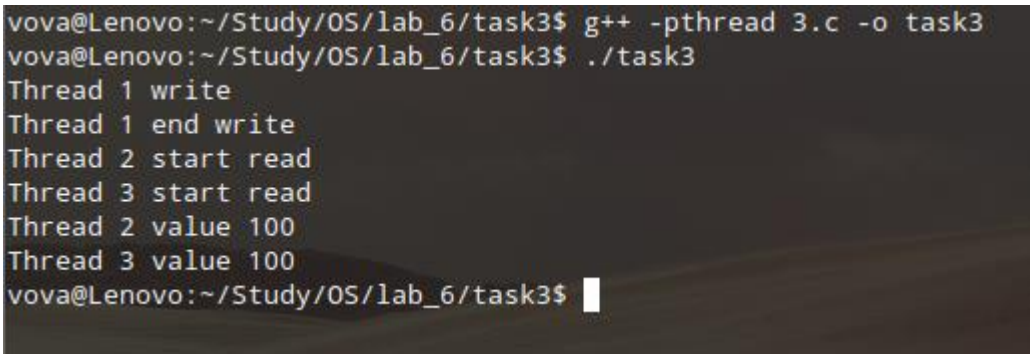
**Завдання 3:**

У цьому завданні потрібно було реалізувати блокування читання-записування з перевагою записування.

Дана програма створює три потоки, два з яких намагаються прочитати змінну someVal, а третій змінити значення цієї змінної. В даній програмі реалізовано механізм читання-записування з перевагою записування, тобто під час запису жоден потік не може працювати зі змінною, окрім записуючого.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>

pthread_rwlock_t rwlock = PTHREAD_RWLOCK_INITIALIZER;
int someVal = 0;

pthread_mutex_t indexMutex = PTHREAD_MUTEX_INITIALIZER
int index = 0;
int getIndex(){
    index++;
    return index;
}

void* readThread(void* p) {
    pthread_mutex_lock(&indexMutex);
    int index = getIndex();
    pthread_mutex_unlock(&indexMutex);

    pthread_rwlock_rdlock(&rwlock);
    printf("Thread %i start read\n", index);
    usleep(500);
    printf("Thread %i value %i\n", index, someVal);
    pthread_rwlock_unlock(&rwlock);
}

void* writeThread(void* p){
    pthread_mutex_lock(&indexMutex);
    int index = getIndex();
    pthread_mutex_unlock(&indexMutex);

    pthread_rwlock_wrlock(&rwlock);
    printf("Thread %i write\n", index);
    usleep(500);
    someVal = index * 100;
    usleep(500);
    printf("Thread %i end write\n", index);
```

Приклад роботи програми:




**Висновок:** на цій лабораторній роботі я ознайомився з принципами міжпотокової взаємодії та навчився реалізовувати їх на практиці.