

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 лабораторної роботи
«Програмна реалізація міжпоточної взаємодії в ОС Windows і Linux»

Виконала:
студентка групи ФЕІ-23
Лісова С.О.
Перевірив:
Сінькевич О.О.

Мета: освоєння методів реалізації міжпоточної взаємодії.

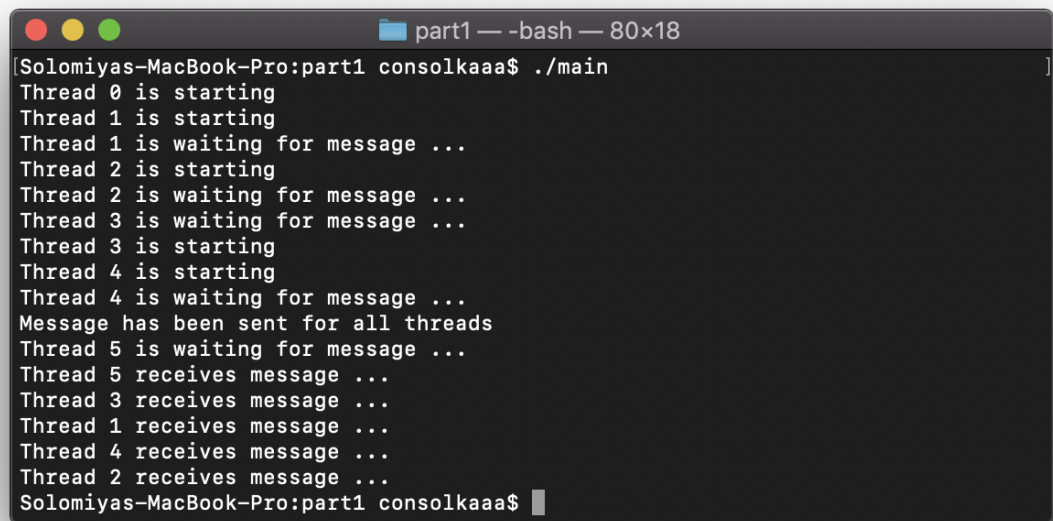
Завдання №1:

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.

```
1. #include <stdio.h>
2. #include <pthread.h>
3.
4. pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
5. pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
6. int condition = 0;
7.
8. void send_msg()
9. {
10.     condition = 1;
11.     pthread_cond_broadcast(&cond);
12. }
13.
14. void resive_msg()
15. {
16.     pthread_mutex_lock(&mutex);
17.     while(!condition)
18.         pthread_cond_wait(&cond, &mutex);
19.     pthread_mutex_unlock(&mutex);
20. }
21.
22. void* someThread(void* p)
23. {
24.     int threadID = *(int*) p;
25.     printf("Thread %i is waiting for message ...\n", threadID);
26.     resive_msg();
27.     printf("Thread %i receives message ...\n", threadID);
28. }
29.
30. int main()
31. {
32.     int THREAD_CONUT = 5;
33.     pthread_t threads[THREAD_CONUT];
34.
35.     for(int i = 0; i < THREAD_CONUT; ++i)
36.     {
37.         printf("Thread %i is starting\n", i);
38.         pthread_create(&threads[i], NULL, someThread, &i);
39.     }
40.
41.     printf("Message has been sent for all threads\n");
42.     send_msg();
43.
44.     for(int i = 0; i < THREAD_CONUT; ++i)
45.         pthread_join(threads[i], NULL);
46.
47.     return 0;
```

48. }

A terminal window titled 'part1 — -bash — 80x18' showing the output of a program. The output lists the start and wait states of threads 0 through 5, followed by a message being sent to all threads and then the receipt of the message by threads 5, 3, 1, 4, and 2 in that order.

```
Solomiyas-MacBook-Pro:part1 consolkaaa$ ./main
Thread 0 is starting
Thread 1 is starting
Thread 1 is waiting for message ...
Thread 2 is starting
Thread 2 is waiting for message ...
Thread 3 is waiting for message ...
Thread 3 is starting
Thread 4 is starting
Thread 4 is waiting for message ...
Message has been sent for all threads
Thread 5 is waiting for message ...
Thread 5 receives message ...
Thread 3 receives message ...
Thread 1 receives message ...
Thread 4 receives message ...
Thread 2 receives message ...
Solomiyas-MacBook-Pro:part1 consolkaaa$
```

Завдання №2:

Реалізуйте спільно використововувану 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>

//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 (stck 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);

        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;
}

```

```

Pushed 1
Pushed 11
Pushed 111
Pushed 1111
Pushed 11111
Pop 11111
Pop 1111
Pop 111
Pop 11
Pop 1
Pushed 2
Pushed 22
Pop 3
Pop 222
Pushed 222
Pushed 2222
Pushed 22222
Pop 22222
Pop 2222
Pop 22
Pop 2
Pop 1
Stack is empty!
Pushed 3
Pushed 33
Pushed 333
Pushed 3333
Pushed 33333
Pop 33333
Pop 3333
Pop 333
Pop 33
Stack is empty!
Stack is empty!

```

Завдання №3:

Розробіть програму реалізації блокувань читання-записування з перевагою за-писування.

```

#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 undex=0;
int getIndex()
{
    undex++;
    return undex;
}

```

```

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 change 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 stop writing\n", index);
    pthread_rwlock_unlock(&rwlock);
}

int main()
{
    pthread_t t1,t2,t3;
    pthread_create(&t1, NULL, readThread, NULL);
    pthread_create(&t2, NULL, writeThread, NULL);
    pthread_create(&t3, NULL, readThread, NULL);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    pthread_join(t3, NULL);
    return 0;
}

```

```

Lab_6 — -bash — 80x80
[Solomiyas-MacBook-Pro:lab_6 consolkaaa$ ./part3
Thread 1 start read
Thread 2 start read
Thread 1 value 0
Thread 2 value 0
Thread 3 write
Thread 3 end write
Solomiyas-MacBook-Pro:lab_6 consolkaaa$

```

Висновок: під час виконання лабораторної роботи я освоїла методи міжпоточної взаємодії.