

Міністерство освіти та науки України  
Львівський національний університет ім. Івана Франка  
Факультет електроніки та комп'ютерних технологій  
Кафедра радіоелектронних і комп'ютерних систем

## ЗВІТ

про виконання лабораторної роботи №6  
«Програмна реалізація міжпотоквої  
взаємодії в ОС Windows і Linux»

Виконав:  
Студент групи ФЕІ-23  
Речинський О.С.  
Перевірив:  
ас. Сінькевич О.О.

Львів – 2019

## Додаткова інформація

Варіант: №6;  
Дистрибутив: Ubuntu 19.04;  
Девайс: Acer Aspire 5 A515-51G-58BE;  
Процесор: Intel Core i5-8250U;  
Графіка: NVIDIA GeForce MX130;  
Оперативна пам'ять: 8 ГБ;  
Постійна пам'ять: 1 ТБ HDD.

### Лабораторна робота №6. Програмна реалізація міжпоточної взаємодії в ОС Windows і Linux

#### Мета

Ознайомитись з механізмами міжпоточної взаємодії.

#### Завдання № 1

Напишіть код таких функцій:

1. `send_msg()`, що відсилає повідомлення і N потокам і припиняє поточний потік, поки усі вони не одержать повідомлення;
2. `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.

#### Результат

```
kick28@kick28-Aspire-A515-51G:~/Desktop$ gcc -pthread -o untitled1 untitled1.c
kick28@kick28-Aspire-A515-51G:~/Desktop$ ./untitled1
Thread 0 is starting
Thread 1 is starting
Thread 2 is starting
Thread 2 is waitin' for msg ...
Thread 2 is waitin' for msg ...
Thread 3 is starting
Thread 3 is waitin' for msg ...
Thread 4 is starting
Thread 4 is waitin' for msg ...
Message sent for all threads
Thread 5 is waitin' for msg ...
Thread 2 got msg ...
Thread 3 got msg ...
Thread 4 got msg ...
Thread 5 got msg ...
Thread 2 got msg ...
```

#### Код програми

```

#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 waitin' for msg ...\n", threadID);
    resive_msg();
    printf("Thread %i got msg ...\n", threadID);
}

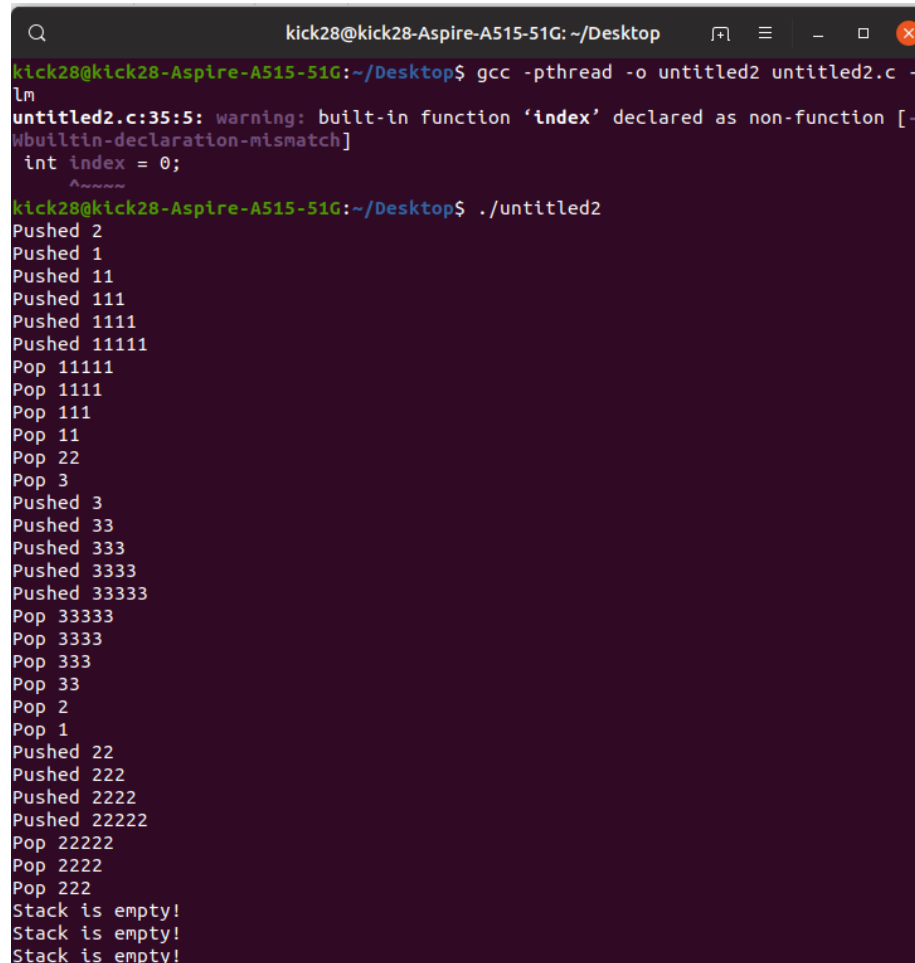
int main()
{
    int THREAD_CONUT = 5;
    pthread_t threads[THREAD_CONUT];
    for(int i = 0; i < THREAD_CONUT; ++i)
    {
        printf("Thread %i is starting\n", i);
        pthread_create(&threads[i], NULL, someThread, &i);
    }
    printf("Message sent for all threads\n");
    send_msg();
    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.

### Результат



```
kick28@kick28-Aspire-A515-51G: ~/Desktop
kick28@kick28-Aspire-A515-51G:~/Desktop$ gcc -pthread -o untitled2 untitled2.c -lm
untitled2.c:35:5: warning: built-in function 'index' declared as non-function [-Wbuiltin-declaration-mismatch]
    int index = 0;
    ^~~~~~
kick28@kick28-Aspire-A515-51G:~/Desktop$ ./untitled2
Pushed 2
Pushed 1
Pushed 11
Pushed 111
Pushed 1111
Pushed 11111
Pop 11111
Pop 1111
Pop 111
Pop 11
Pop 22
Pop 3
Pushed 3
Pushed 33
Pushed 333
Pushed 3333
Pushed 33333
Pop 33333
Pop 3333
Pop 333
Pop 33
Pop 2
Pop 1
Pushed 22
Pushed 222
Pushed 2222
Pushed 22222
Pop 22222
Pop 2222
Pop 222
Stack is empty!
Stack is empty!
Stack is empty!
```

### Код програми

```
#include <stdio.h>
#include <pthread.h>
#include <math.h>
#include <stdlib.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 = malloc(size * sizeof(int));
}
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)

```

```

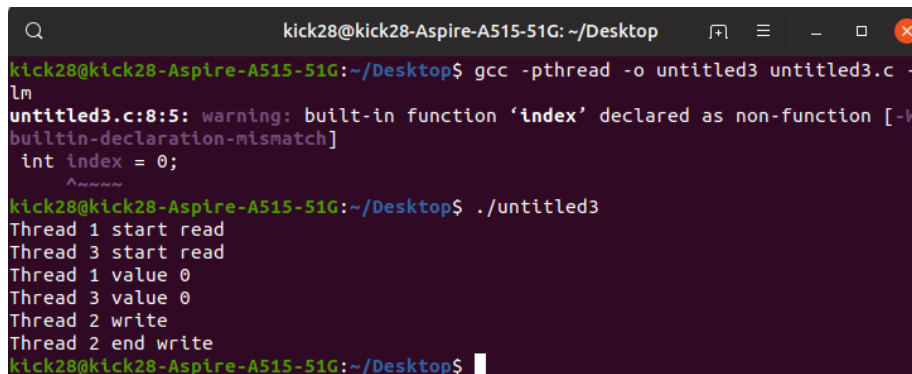
if ((currentVal = pop()) != -1)
printf("Pop %i \n", currentVal);
else
printf("Stack is empty!\n");
}
int main() {
int THREAD_COUNT = 3;
pthread_t threads[THREAD_COUNT];
createStack(THREAD_COUNT * 5);
for (int i = 1; i <= THREAD_COUNT; ++i) {
pthread_create(&threads[i - 1], NULL, threadFunc, NULL);
}
for (int i = 0; i < THREAD_COUNT; ++i)
pthread_join(threads[i], NULL);
return 0;
}

```

### Завдання № 3

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

### Результат



```

kick28@kick28-Aspire-A515-51G: ~/Desktop
kick28@kick28-Aspire-A515-51G:~/Desktop$ gcc -pthread -o untitled3 untitled3.c -lm
untitled3.c:8:5: warning: built-in function 'index' declared as non-function [-Wbuiltin-declaration-mismatch]
    int index = 0;
    ^~~~~~
kick28@kick28-Aspire-A515-51G:~/Desktop$ ./untitled3
Thread 1 start read
Thread 3 start read
Thread 1 value 0
Thread 3 value 0
Thread 2 write
Thread 2 end write
kick28@kick28-Aspire-A515-51G:~/Desktop$

```

### Код програми

```

#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);
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;
}

```

### Висновок

На цій лабораторній роботі я ознайомився з принципами міжпоточної взаємодії та навчився реалізовувати їх на практиці.