

Львівський національний університет імені Івана Франка
Факультет електроніки та комп'ютерних технологій

Звіт
про виконання лабораторної роботи №6
“Програмна реалізація міжпоточної взаємодії в ОС Windows і Linux”

Виконав
Студент групи ФЕП-13
Андріан Карсанашвілі
Перевірив:
доц. Бойко. Я. В.

Львів – 2021

Мета: ознайомитись з механізмами міжпоточної взаємодії.

Код 1:

```
#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 reseive_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);
    reseive_msg();
    printf("Thread %i reseived the message ...\n", threadID);
}

int main(){
    int THREAD_COUNT=5;
    pthread_t threads[THREAD_COUNT];

    for (int i=0;i<THREAD_COUNT; 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_COUNT; i++)
        pthread_join(threads[i], NULL);
    return 0;
}
```

```
Thread 0 is starting
Thread 1 is starting
Thread 2 is starting
Thread 2 is waiting for message ...
Thread 3 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 sent for all threads
Thread 5 is waiting for message ...
Thread 2 reseived the message ...
Thread 3 reseived the message ...
Thread 3 reseived the message ...
Thread 4 reseived the message ...
Thread 5 reseived the message ...
```

Код 2:

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

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){
    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 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 %1\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=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 2
Pop 1111
Pop 111
Pop 11
Pop 1
Pushed 2
Pushed 22
Pushed 222
Pushed 2222
Pushed 22222
Pop 22222
Pop 2222
Pop 222
Pop 22
Stack is empty!
Stack is empty!
Pushed 3
Pushed 33
Pushed 333
Pushed 3333
Pushed 33333
Pop 33333
Pop 3333
Pop 333
Pop 33
Pop 3
Stack is empty!

```

Код 3:

```
#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;
}
```

```
Thread 1 start read  
Thread 3 start read  
Thread 1 value 0  
Thread 3 value 0  
Thread 2 write  
Thread 2 end write
```

Висновок : на даній лабораторній роботі я навчився застосовувати механізми міжпоточної взаємодії між процесами. Вивчив основний інструментарій виконання роботи потоків.