НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-ОБЧИСЛЮВАЛЬНИХ ТЕХНОЛОГІЙ

КАФЕДРА АВТОМАТИКИ ТА УПРАВЛІННЯ В ТЕХНІЧНИХ СИСТЕМАХ

Лабораторна робота № 3

з предмету «Програмне забезпечення комп’ютерних систем»

на тему «ПРАКТИЧНЕ ЗНАЙОМСТВО З ПОТОКАМИ І СИНХРОНІЗАЦІЇ ПОТОКІВ В ОС UNIX»

Виконала:

студентка групи ІТ-з03мп

Козак О.С.

Перевірив:

Мітін С. В.

Київ – 2020

**Лабораторна робота № 3.  
«ПРАКТИЧНЕ ЗНАЙОМСТВО З ПОТОКАМИ І СИНХРОНІЗАЦІЇ ПОТОКІВ В ОС UNIX»**

**Мета**: Ознайомитись з потоками і синхронізацією потоків в ОС UNIX.

**Хід роботи:**

**2-3**. 1) Створюю багатопоточний застосуноку з синхронизацією потоків за допомогою м'ютекса.

Лістинг (два потоки збільшують значення загальної змінної):

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <iostream>

#define INCREMENT\_CYCLES 1000000

**using** **namespace** std**;**

int counter**;**

pthread\_mutex\_t mutex**;**

void**\*** threadRoutine**(**void**\*** arg**);**

int main**()**

**{**

pthread\_t thread1**,** thread2**;**

pthread\_mutex\_init**(&**mutex**,** **NULL);**

pthread\_create**(&**thread1**,** **NULL,** **&**threadRoutine**,** **NULL);**

pthread\_create**(&**thread2**,** **NULL,** **&**threadRoutine**,** **NULL);**

pthread\_join**(**thread1**,** **NULL);**

pthread\_join**(**thread2**,** **NULL);**

pthread\_mutex\_destroy**(&**mutex**);**

cout **<<** "Expected counter value: " **<<** INCREMENT\_CYCLES **\*** 2 **<<** endl**;**

cout **<<** "Actual counter value: " **<<** counter **<<** endl**;**

**}**

void**\*** threadRoutine**(**void**\*** arg**)**

**{**

pthread\_t executingThread **=** pthread\_self**();**

cout **<<** "Thread " **<<** executingThread **<<** ": Started." **<<** endl**;**

**for** **(**int i **=** 0**;** i **<** INCREMENT\_CYCLES**;** i**++)**

**{**

pthread\_mutex\_lock**(&**mutex**);**

counter **+=** 1**;**

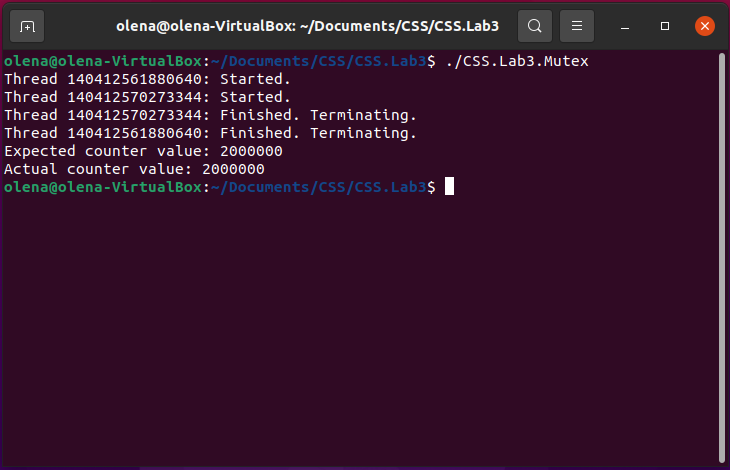
pthread\_mutex\_unlock**(&**mutex**);**

**}**

cout **<<** "Thread " **<<** executingThread **<<** ": Finished. Terminating." **<<** endl**;**

**}**

Результат виконання програми:

****

**2)** Створюю багатопоточний застосунок з синхронизацією потоків за допомогою семафору.

Лістинг (три потоки виконують критичну секцію коду; семафором обмежена кількість потоків які можуть одночасно виконувати код критичної секції (до 2х)):

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

#include <iostream>

**using** **namespace** std**;**

sem\_t semaphore**;**

void **\***threadRoutine**(**void **\***arg**);**

int main**()**

**{**

sem\_init**(&**semaphore**,** 0**,** 2**);**

pthread\_t th1**,** th2**,** th3**;**

// Create threads

pthread\_create**(&**th1**,** **NULL,** threadRoutine**,** **NULL);**

usleep**(**600000**);**

pthread\_create**(&**th2**,** **NULL,** threadRoutine**,** **NULL);**

usleep**(**600000**);**

pthread\_create**(&**th3**,** **NULL,** threadRoutine**,** **NULL);**

// Join threads

pthread\_join**(**th1**,** **NULL);**

pthread\_join**(**th2**,** **NULL);**

pthread\_join**(**th3**,** **NULL);**

sem\_destroy**(&**semaphore**);**

**}**

void **\***threadRoutine**(**void **\***arg**)**

**{**

pthread\_t executingThread **=** pthread\_self**();**

cout << "Thread " << executingThread << ": Started." << endl;

cout << "Thread " << executingThread << ": Waiting on semaphore..." << endl;

sem\_wait(&semaphore);

cout << "Thread " << executingThread << ": Acquired semaphore." << endl;

sleep(1);

cout << "Thread " << executingThread << ": Work done." << endl;

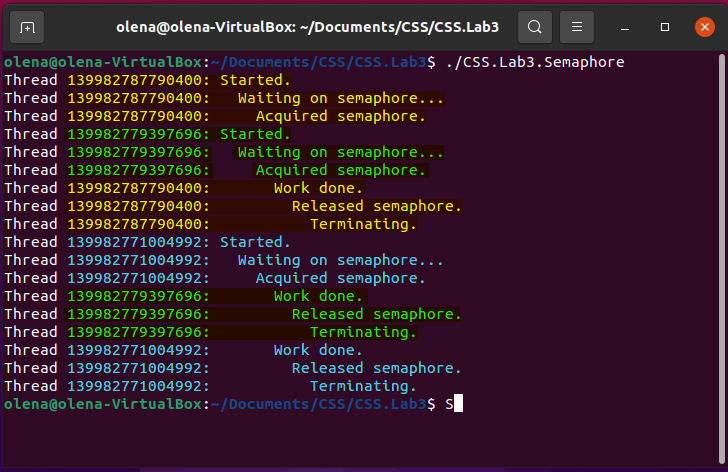
sem\_post(&semaphore);

cout << "Thread " << executingThread << ": Released semaphore." << endl;

cout << "Thread " << executingThread << ": Terminating." << endl;

}

Результат виконання програми:

****

3) Створюю багатопоточний застосунок з синхронизацією потоків за допомогою умовної змінної.

Лістинг (два потоки з випадковим інтервалом збільшують значення загальної змінної; третій поток обнуляє її, як тільки її значення досягає певної відмітки):

#include <stdlib.h>

#include <stdio.h>

#include <iostream>

#include <unistd.h>

#include <pthread.h>

#define CONSUMPTION\_MIN\_PORTION 3

#define PRODUCTION\_GOAL 15

**using** **namespace** std**;**

int productsAvailable **=** 0**;**

int totalProductsProduced **=** 0**;**

pthread\_mutex\_t mutex**;**

pthread\_cond\_t condition**;**

void**\*** producerThreadRoutine**(**void **\***args**);**

void**\*** consumerThreadRoutine**(**void **\***args**);**

int main**(**int argc**,** char **\***argv**[])**

**{**

pthread\_t thProducer1**,** thProducer2**,** thConsumer**;**

pthread\_mutex\_init**(&**mutex**,** **NULL);**

pthread\_cond\_init**(&**condition**,** **NULL);**

srand**((**unsigned**)** time**(NULL)** **\*** getpid**());**

pthread\_create**(&**thProducer1**,** **NULL,** producerThreadRoutine**,** **NULL);**

usleep**(**500000**);**

pthread\_create**(&**thProducer2**,** **NULL,** producerThreadRoutine**,** **NULL);**

pthread\_create**(&**thConsumer**,** **NULL,** consumerThreadRoutine**,** **NULL);**

pthread\_join**(**thProducer1**,** **NULL);**

pthread\_join**(**thProducer2**,** **NULL);**

pthread\_join**(**thConsumer**,** **NULL);**

**}**

void**\*** consumerThreadRoutine**(**void **\***args**)**

**{**

pthread\_t executingThread **=** pthread\_self**();**

cout **<<** " Consumer thread " **<<** executingThread **<<** ": Started." **<<** endl**;**

**while(**totalProductsProduced **<** PRODUCTION\_GOAL**)**

**{**

pthread\_mutex\_lock**(&**mutex**);**

**while** **(**productsAvailable **<** CONSUMPTION\_MIN\_PORTION**)**

**{**

pthread\_cond\_wait**(&**condition**,** **&**mutex**);**

**}**

cout **<<** " Consumer thread " **<<** executingThread **<<** ": Consuming available products..." **<<** endl**;**

productsAvailable **=** 0**;**

cout **<<** " Consumer thread " **<<** executingThread **<<** ": Products available: 0" **<<** endl**;**

pthread\_mutex\_unlock**(&**mutex**);**

**}**

**}**

void**\*** producerThreadRoutine**(**void **\***args**)**

**{**

pthread\_t executingThread **=** pthread\_self**();**

cout **<<** "Producer thread " **<<** executingThread **<<** ": Started." **<<** endl**;**

**while** **(**totalProductsProduced **<** PRODUCTION\_GOAL**)**

**{**

pthread\_mutex\_lock**(&**mutex**);**

productsAvailable**++;**

totalProductsProduced**++;**

cout **<<** "Producer thread " **<<** executingThread **<<** ": Produced 1 item. "**;**

cout **<<** "Products available: " **<<** productsAvailable **<<** endl**;**

**if** **(**productsAvailable **>=** CONSUMPTION\_MIN\_PORTION**)**

**{**

pthread\_cond\_signal**(&**condition**);**

**}**

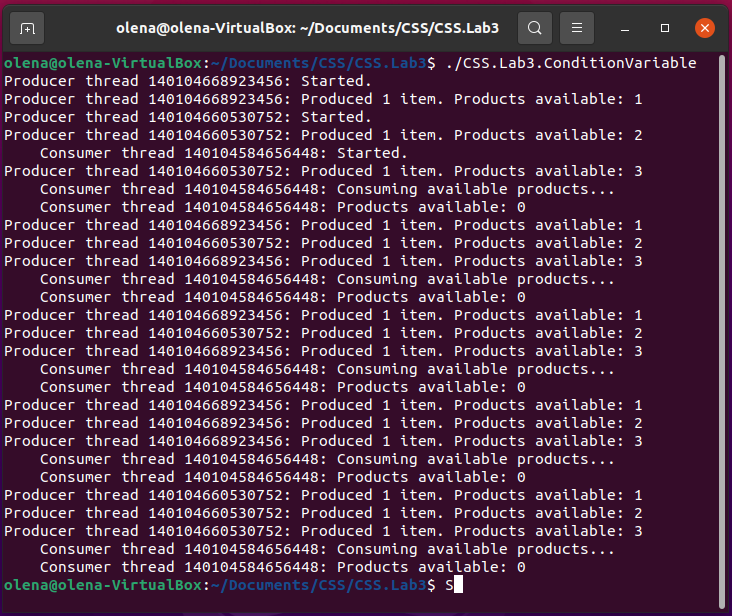
pthread\_mutex\_unlock**(&**mutex**);**

usleep**(**rand**()** **%** 3000000**);**

**}**

**}**

Результат виконання програми:

**4.** Розглянемо приклад коли синхронізаціїєю потоків буде знехтувано на прикладі застосунку.

Лістинг (код, що відповідає за синхронізацію потоків закоментованю):

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <iostream>

#define INCREMENT\_CYCLES 1000000

**using** **namespace** std**;**

int counter**;**

//pthread\_mutex\_t mutex;

void**\*** threadRoutine**(**void**\*** arg**);**

int main**()**

**{**

pthread\_t thread1**,** thread2**;**

//pthread\_mutex\_init(&mutex, NULL);

pthread\_create**(&**thread1**,** **NULL,** **&**threadRoutine**,** **NULL);**

pthread\_create**(&**thread2**,** **NULL,** **&**threadRoutine**,** **NULL);**

pthread\_join**(**thread1**,** **NULL);**

pthread\_join**(**thread2**,** **NULL);**

//pthread\_mutex\_destroy(&mutex);

cout **<<** "Expected counter value: " **<<** INCREMENT\_CYCLES **\*** 2 **<<** endl**;**

cout **<<** "Actual counter value: " **<<** counter **<<** endl**;**

**}**

void**\*** threadRoutine**(**void**\*** arg**)**

**{**

pthread\_t executingThread **=** pthread\_self**();**

cout **<<** "Thread " **<<** executingThread **<<** ": Started." **<<** endl**;**

**for** **(**int i **=** 0**;** i **<** INCREMENT\_CYCLES**;** i**++)**

**{**

//pthread\_mutex\_lock(&mutex);

counter **+=** 1**;**

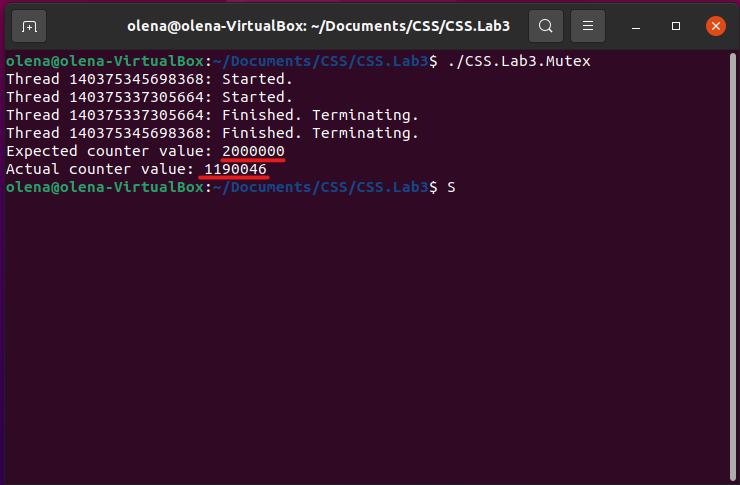
//pthread\_mutex\_unlock(&mutex);

**}**

cout **<<** "Thread " **<<** executingThread **<<** ": Finished. Terminating." **<<** endl**;**

**}**

Результат виконання програми:

Пояснення отриманого результату.

Коли доступ до загальної змінної кількома потокам не синхронізований відбувається так званий стан гонитви (race condition), адже потоки одночасно зчитують і оновлюють значення змінної без синхронізації.   
Приклад:

1. Поток 1 зчитує значення (отримав 0)
2. Поток 2 зчитує значення (отримав 0)
3. Поток 1 розрахував нове значення (0+1) і записав
4. Поток 2 розрахував нове значення (0+1) і записав

Таким чином, хоч і було виконано 2000000 інкременті, але через перезаписування і несвоєчасну вичитку, було втрачено майже половину інкрементів.

**Висновок:**

Під час виконання лабораторної роботи я ознайомилась з потоками і синхронізацією потоків в ОС UNIX.