ESE 2025 Thread Report

Instructor: Takis Zourntos

Student: Vy Nguyen - C0776242

Introduction

This report introduces the "pthread" library with two examples on some of the function, These examples can be found on: https://github.com/takiszourntos/teaching/tree/master/lambton/2020/summer/ese2025/week_11/workspace

Discussion

In order to run each of the programs, the library pthread should be linked to the selected project by using these steps:

- Project - > Properties -> C/C++ Build -> Settings -> Tool Settings tabs ->GCC C++ Linker ->Libraries -> add "pthread".

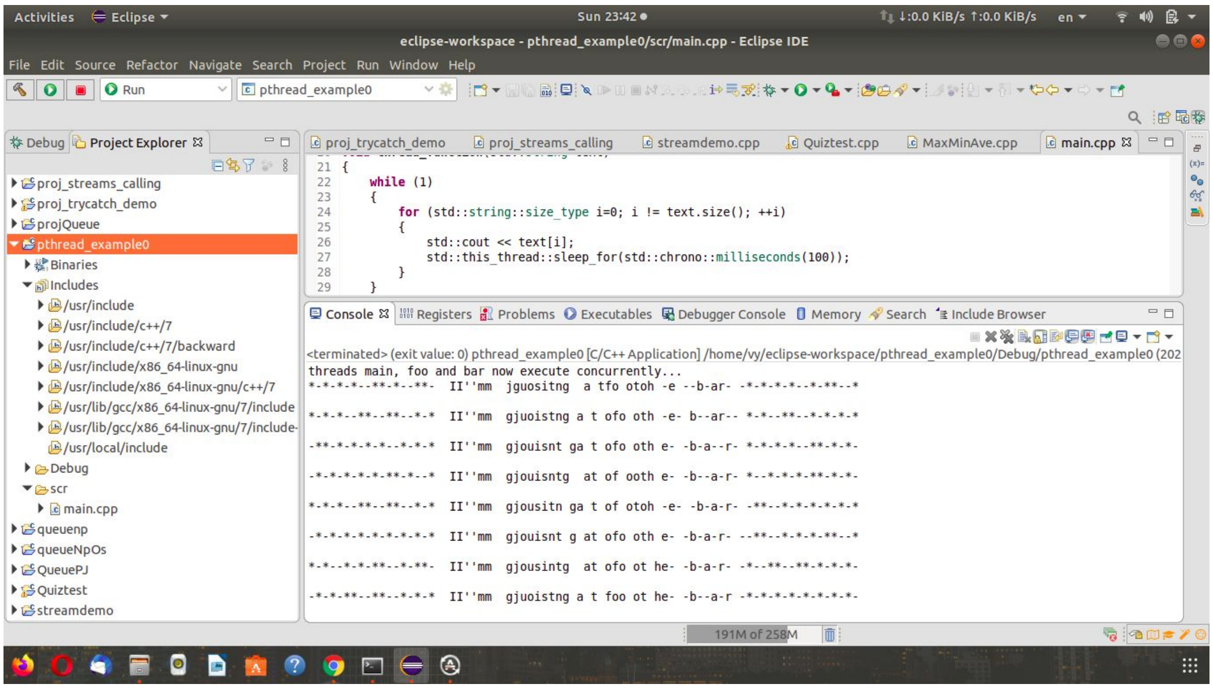
Then

- Project - > Properties -> C/C++ Build -> Settings -> Tool Settings tabs ->GCC C++ Linker -> Miscellaneous -> add "-pthread".

Running the programs:

Example 0:

With the code provided by professor Takis Zourntos, the output of the program is:



```
{
    while (1)
    {
        for (std::string::size_type i=0; i != text.size(); ++i)
        {
            std::cout << text[i];
            std::this_thread::sleep_for(std::chrono::milliseconds(100));
        }
    }
```

```
<terminated> (exit value: 0) pthread_example0 [C/C++ Application] /home/vy/eclipse-workspace/pthread_example0/Debug/pthread_example0 (202
threads main, foo and bar now execute concurrently...
*-*-*-*--**-*--**-  II''mm  jguositng  a tfo otoh -e --b-ar- -*-*-*-*--*-**--*

*-*-*--**--**--*-*  II''mm  gjuoistng a t ofo oth -e- b--ar-- *-*--**--*-*-*-*

-**-*-*-*-*--*-*-*  II''mm  gjouisnt ga t ofo oth e- -b-a--r- *-*-*-*--**-*-*-

-*-*-*-*-*-**-*--*  II''mm  gjouisntg  at of ooth e- -b--a-r- *--*-*-*-**-*-*-

*-*-*--**--**--*-*  II''mm  gjousitn ga t of otoh -e- -b-a-r- -**--*-*-*-*-*-*

-*-*-*-*-*-*-*-*-*  II''mm  gjouisnt g at ofo oth e- -b-a-r- --**--*-*-*-**--*

*-*--*-*-**--*-**-  II''mm  gjousintg  at ofo ot he- -b-a-r- -*--**--**-*-*-*-

-*-*-**--**--*-*-*  II''mm  gjuoistng a t foo ot he- -b--a-r -*-*-*-*-*-*-*-*-
```

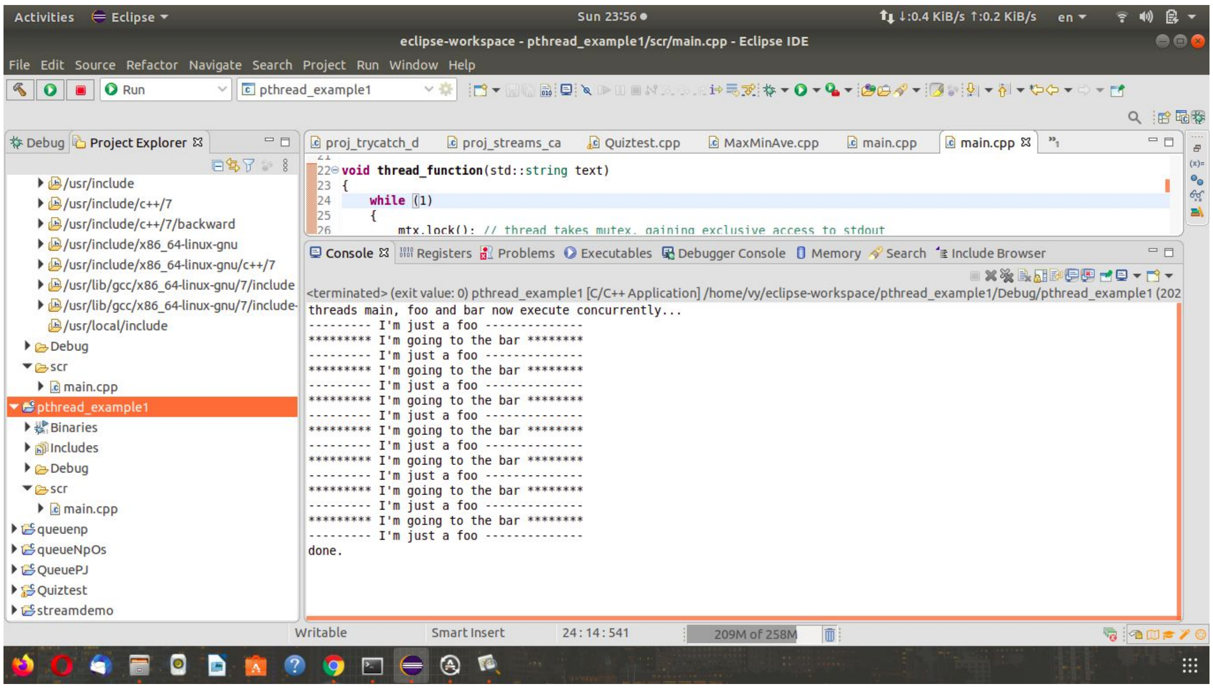
There are some points in the programs should be noticed:

1. There is a delay of 0.1s (100 milliseconds) between each output line.
2. The two threads (strings) are messed up.

Functions should be noted:

```
std::this_thread::sleep_for(std::chrono::milliseconds(100));
detach();
```

In example 1, which is a better version of example 0, we have the result:



```
void thread_function(std::string text)
{
    while (1)
    {
        mtx.lock(); // thread takes mutex, gaining exclusive access to stdout
```

```
<terminated> (exit value: 0) pthread_example1 [C/C++ Application] /home/vy/eclipse-workspace/pthread_example1/Debug/pthread_example1 (202
threads main, foo and bar now execute concurrently...
--------- I'm just a foo -------------
********* I'm going to the bar ********
--------- I'm just a foo -------------
********* I'm going to the bar ********
--------- I'm just a foo -------------
********* I'm going to the bar ********
--------- I'm just a foo -------------
********* I'm going to the bar ********
--------- I'm just a foo -------------
********* I'm going to the bar ********
--------- I'm just a foo -------------
********* I'm going to the bar ********
--------- I'm just a foo -------------
********* I'm going to the bar ********
--------- I'm just a foo -------------
done.
```

Notice that the output are not messing up like the previous one, so how it can be better?
Answer: With the help of mutex library.
in thread_function, in the while loop, we can include:

```
mtx.lock(); // thread takes mutex, gaining exclusive access to stdout
```

and

```
mtx.unlock(); // release mutex to allow other threads to access stdout
std::this_thread::sleep_for(std::chrono::milliseconds(10)); // wait to be fair
```

Explain for mutex lock and unlock:
A mutex is a lockable_object that is designed to signal when critical sections of code need exclusive access, preventing other threads with the same protection from executing concurrently and access the same memory locations.
(source: http://www.cplusplus.com/reference/mutex/mutex/)
So each thread will take turns to process its content

- If the mutex isn't currently locked by any thread, the calling thread locks it (from this point, and until its member unlock is called, the thread owns the mutex).
- If the mutex is currently locked by another thread, execution of the calling thread is blocked until unlocked by the other thread (other non-locked threads continue their execution).
- If the mutex is currently locked by the same thread calling this function, it produces a deadlock (with undefined behavior). See recursieve_mute for a mutex type that allows multiple locks from the same thread.

Summary:
This is just some introduction steps to get familiar with the thread library in C++. and the help of mutex to prevent corruption in the outputs of the threads.

References:

http://www.cplusplus.com/reference/mutex/mutex/
http://www.cplusplus.com/reference/mutex/mutex/lock/

Appendix
Example 0:

```
/*
 * main.cpp
 *
 *  Created on: Aug. 10, 2020
 *
 *  Two threads created from one thread function
 *
 *      NB: to build, you should indicate the "pthread" library for your C++ linker
 *
 *
 *  author: T. Zourntos
 *
 */

// thread example
#include <iostream>          // std::cout
#include <thread>            // std::thread
#include <chrono>            // std::chrono

void thread_function(std::string text)
{
       while (1)
       {
              for (std::string::size_type i=0; i != text.size(); ++i)
              {
                     std::cout << text[i];
                     std::this_thread::sleep_for(std::chrono::milliseconds(100));
              }
       }
}

int main()
{
       // spawn new thread called foo
       std::thread foo(thread_function,
                     "--------- I'm just a foo --------------\n");

       // spawn new thread called bar
       std::thread bar(thread_function,
                     "********* I'm going to the bar ********\n");

       // detach threads to allow "safe" termination
       foo.detach();
       bar.detach();

       // send status message and wait
       std::cout << "threads main, foo and bar now execute concurrently...\n";
       std::this_thread::sleep_for(std::chrono::seconds(60));

       // end proceedings...
       std::cout << "done.\n";
```

```
        return 0;
}
```

Example 1:

```
/*
 * main.cpp
 *
 *  Created on: Aug. 10, 2020
 *
 *  mutex example
 *
 *     author: T. Zourntos
 *      some inspiration/lines from source: http://www.cpp.re/reference/mutex/mutex/
 *
 *      NB: to build, you should indicate the "pthread" library for your C++ linker
 */

// thread example
#include <iostream>           // std::cout
#include <thread>             // std::thread
#include <chrono>             // std::chrono
#include <mutex>        // std::mutex

std::mutex mtx;           // mutex for critical section

void thread_function(std::string text)
{
        while (1)
        {
                mtx.lock(); // thread takes mutex, gaining exclusive access to stdout

                for (std::string::size_type i = 0; i != text.size(); ++i)
                {
                        std::cout << text[i];
                        std::this_thread::sleep_for(std::chrono::milliseconds(100));
                }

                mtx.unlock(); // release mutex to allow other threads to access stdout
                std::this_thread::sleep_for(std::chrono::milliseconds(10)); // wait to be fair
        }
}

int main()
{
        // spawn new thread called foo
        std::thread foo(thread_function,
                        "--------- I'm just a foo --------------\n");

        // spawn new thread called bar
        std::thread bar(thread_function,
                        "********* I'm going to the bar ********\n");

        // detach threads to allow "safe" termination
        foo.detach();
        bar.detach();

        // send status message and wait
        std::cout << "threads main, foo and bar now execute concurrently...\n";
        std::this_thread::sleep_for(std::chrono::seconds(60));

        // end proceedings...
        std::cout << "done.\n";
        return 0;
}
```