**Portfolio Milestone Assignment Module Seven Assignment**

**Portfolio Project Part 1**

Victor Enogwe

Computer Science, Colorado State University - Global Campus

CSC450-1: Programming III

Reginald Haseltine

May 20, 2024

**Multithreaded Integer Counter Program**

This assignment demonstrates concurrent execution in C++ using threads.

The application executes two counter threads, counting up to twenty and counting down to zero. The following analysis details the various concepts encountered during the program's implementation.

**Performance issues with concurrency**

Concurrency in our counter program uses threads. The program has three threads: the “main” thread, the “count\_to\_twenty” thread, and the “count\_to\_zero” thread. Since memory space is shared between the threads, no special performance issues arise during memory access across the threads.

Concurrent execution means the threads execute based on some time slot scheduling mechanism. Since the process shared by the threads is run on a single processor core, performance is slower than when the thread functions are executed in parallel.

The shared memory space introduces a state management problem, where each thread must ensure that it correctly reads and writes to the shared state exclusively. This necessitates using a mutex lock to protect the global “shared\_state” count variable, ensuring that each thread has exclusive access to read and write to the counter. If this is not implemented, the shared counter will have indeterminate values at the end of the program run.

Many mutex lock types exist; for the sake of this program and simplicity, I have used the default mutex class.

**Vulnerabilities exhibited with the use of strings.**

Strings from an unbounded source, such as “stdin”, will not have their values and size known at compile time. The security risk this poses is defined in the SET Cert C++ standard “STR50-CPP” (Ballman, 2016, p. 200). When not properly stored, such an input can exceed the buffer bounds and allocated variable size.

Since we do not evaluate any program strings directly from an unbounded input, we do not have to worry about injection attacks or perform any string sanitisation.

**Security of the data types exhibited.**

The two data types used in this program are string and integer. For the string data type, there is no externally provided value; the strings used are constants with predefined memory allocations. Hence, the program does not need to validate, sanitise, or account for buffer overflows when storing the strings (Ballman, 2016, p. 200).

Signs and sizes must be accounted for when using integers to avoid an integer overflow (Coker et al., n.d., p.1). We do not need to specify an “int” size type such as “short” since we are going to run this on a 32/64-bit computer, where the “int” type will be converted appropriately.

**Program Pseudocode**:

PROGRAM Thread Counters  
- Multithreaded concurrent program that counts up to twenty and down to zero.  
  
BEGIN  
 BEGIN  
 Define a "count\_to\_twenty" function that counts up to twenty using a loop.  
 The "count\_to\_twenty" takes a single parameter, a reference to a global count variable.  
 Print the result of each increment.  
 END  
  
 BEGIN  
 Define a "count\_to\_zero" function that counts down to zero using a loop.  
 The "count\_to\_zero" takes a single parameter, a reference to a global count variable.  
 Print the result of each decrement.  
 END  
  
 BEGIN  
 Define a "thread\_runner" function that takes the following parameters:  
 - name: the thread's name  
 - thread\_function: the function to execute within the thread  
 - count: a reference to a global count variable  
 - thread\_lock: a mutex lock to control access to the global count variable  
 The "thread\_runner" function should execute the "thread\_function" and control access to the shared count variable.  
 END  
  
 Define a "main" method.  
 The main method should execute the "count\_to\_twenty" function using the "thread\_runner" in a separate thread.  
 The main method should execute the "count\_to\_zero" function using the "thread\_runner" in a separate thread.  
 Both threads should run concurrently, and the "counter" global variable state should be valid.  
END

**thread.h:**

/\*

\* Multithreaded Count to twenty and back to zero program.

\* thread.h

\*

\* Created by Victor Enogwe on 20/05/2024.

\*/

#ifndef THREAD\_H

#define THREAD\_H

#include <thread>

using std::string;

using std::mutex;

void thread\_details(const string &name);

void thread\_runner(

const string &name,

void (&thread\_function)(int&),

int& thread\_count,

mutex &thread\_lock

);

#endif //THREAD\_H

**thread.cpp:**

/\*

\* Multithreaded Count to twenty and back to zero program.

\* thread.cpp

\*

\* Created by Victor Enogwe on 20/05/2024.

\*/

#include <iostream>

#include <thread>

#include <string>

#include <unistd.h>

#include "thread.h"

using std::cout;

using std::mutex;

using std::this\_thread::get\_id;

void thread\_details(const string &name) {

printf("Running %s Process: %d\n", name.c\_str(), getpid());

cout << name << ": " << get\_id() << "\n";

}

void thread\_runner(

const string &name,

void (&thread\_function)(int&),

int& thread\_count,

mutex &thread\_lock

) {

thread\_lock.lock();

thread\_details(name);

thread\_function(thread\_count);

thread\_lock.unlock();

}

**count.h:**

/\*

\* Multithreaded Count to twenty and back to zero program.

\* count.h

\*

\* Created by Victor Enogwe on 20/05/2024.

\*/

#ifndef COUNT\_H

#define COUNT\_H

void count\_to\_twenty(int &thread\_count);

void count\_to\_zero(int &thread\_count);

#endif //COUNT\_H

**count.cpp:**

/\*

\* Multithreaded Count to twenty and back to zero program.

\* count.cpp

\*

\* Created by Victor Enogwe on 20/05/2024.

\*/

#include <iostream>

#include "count.h"

using std::cout;

void count\_to\_twenty(int &thread\_count) {

// critical section reading a shared variable.

cout << "Count From " << thread\_count << " to 20\n";

while (thread\_count <= 20) {

cout << thread\_count << "\n";

thread\_count++; // critical section writing to a shared variable.

}

thread\_count--; // critical section writing to a shared variable.

cout << "\n";

}

void count\_to\_zero(int &thread\_count) {

// critical section reading a shared variable.

cout << "count from " << thread\_count << " to 0\n";

while (thread\_count >= 0) {

cout << thread\_count << "\n";

thread\_count--; // critical section writing to a shared variable.

}

cout << "\n";

}

**main.cpp:**

/\*

\* Multithreaded Count to twenty and back to zero program.

\* main.cpp

\*

\* Created by Victor Enogwe on 20/05/2024.

\*/

#include <iostream>

#include "thread.h"

#include "count.h"

using std::cout;

using std::thread;

using std::ref;

int main() {

thread\_details("Count Up and Down Main Thread");

cout << "\n";

// critical variable modified in two concurrent threads.

int thread\_count = 0;

mutex thread\_lock;

thread count\_up(

thread\_runner,

"Count Up Thread",

ref(count\_to\_twenty),

ref(thread\_count),

ref(thread\_lock)

);

thread count\_down(

thread\_runner,

"Count Down Thread",

ref(count\_to\_zero),

ref(thread\_count),

ref(thread\_lock)

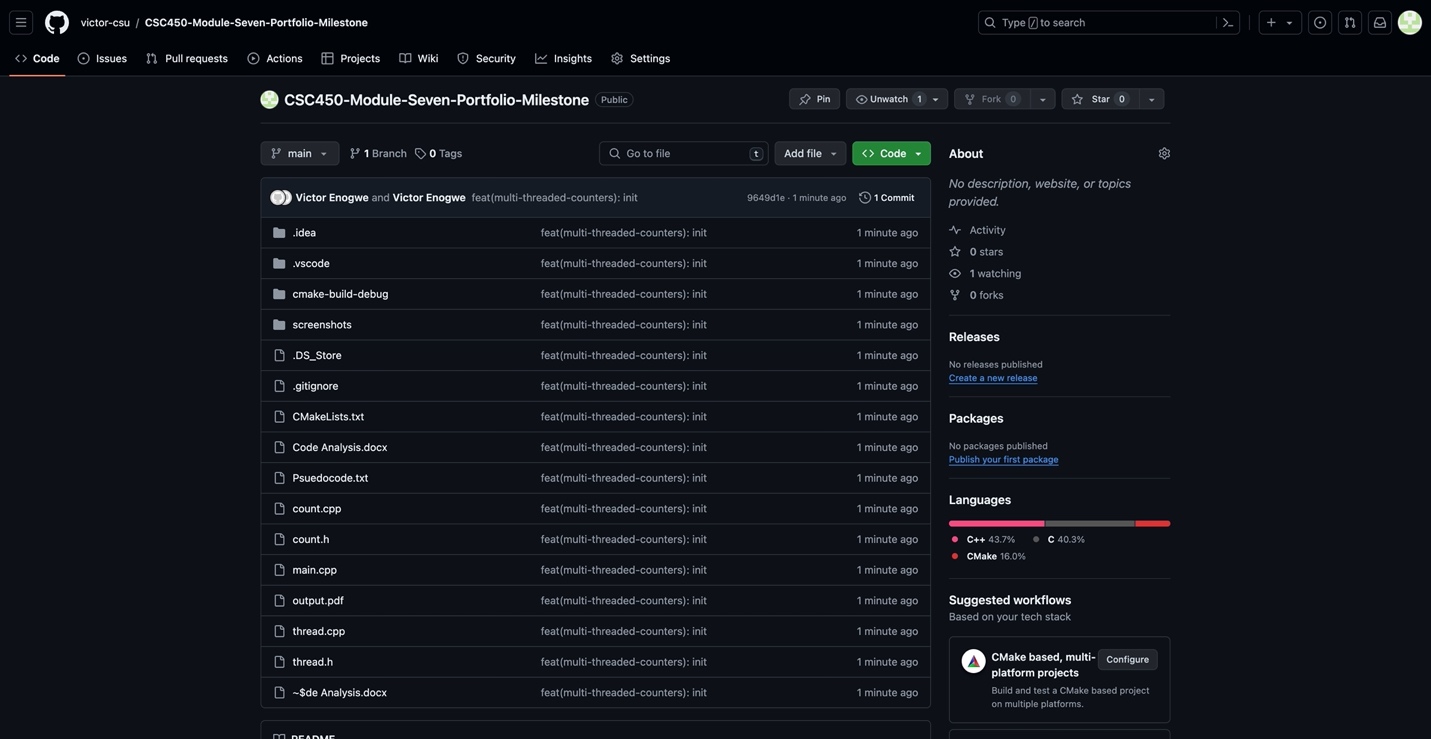
);

count\_up.join();

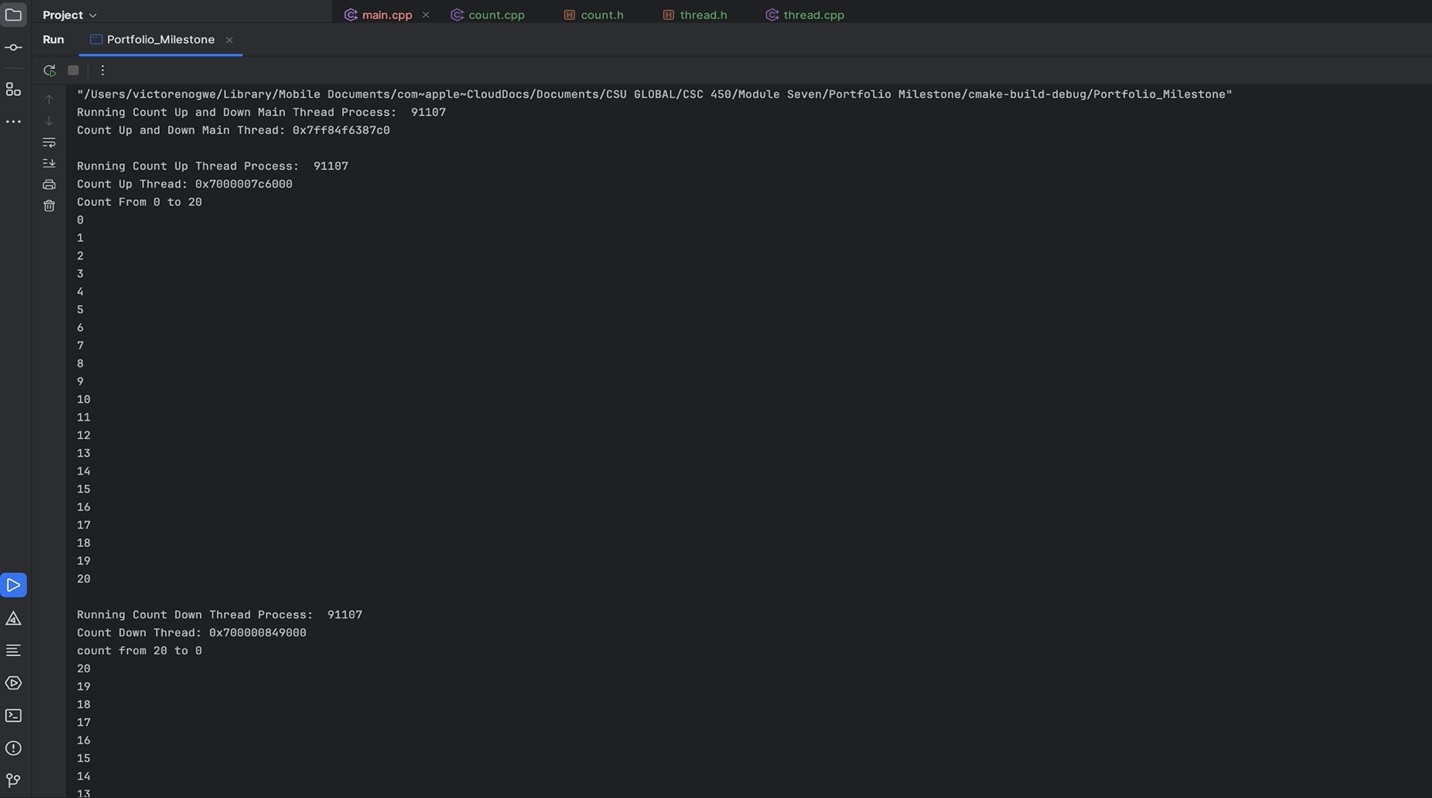
count\_down.join();

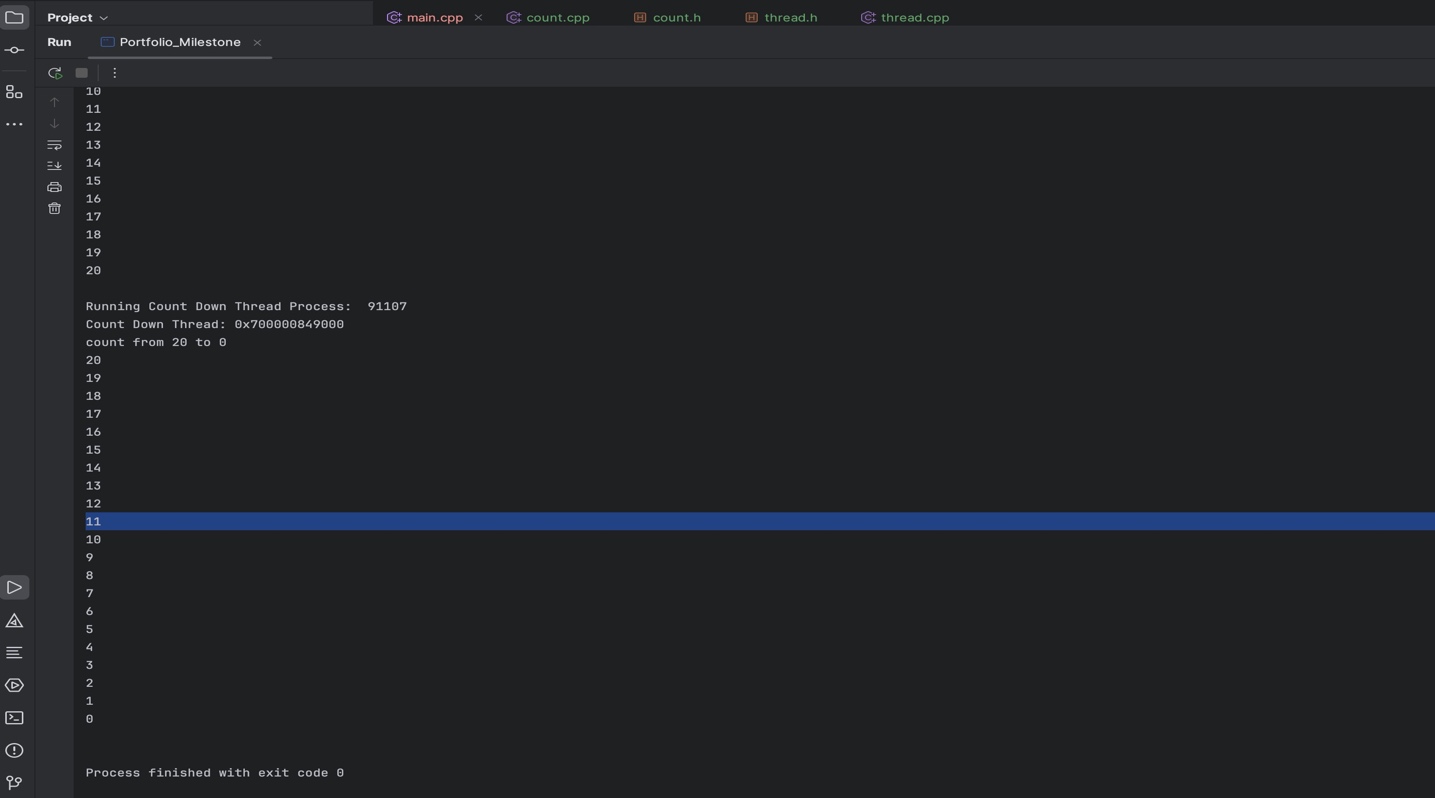
return 0;

}

**Git Repository Image: Git Branch = Main** **-** **<https://github.com/victor-csu/CSC450-Module-Seven-Portfolio-Milestone>**

**Happy Path Execution Screenshot – Fictional Person - CSC450\_PP1\_mod7-execution-output:**





References

Coker, Z., Hassan, S., Overbey, J., Hafiz, M., & Kästner†, C. (n.d.). Integers In C: An Open Invitation To Security Attacks? <https://www.cs.cmu.edu/~ckaestne/pdf/csse14-01.pdf>

Ballman, A. (2016). *SEI CERT C++ Coding Standard: Rules for Developing Safe, Reliable, and Secure Systems in C++* (V01-20170309-0910). Rules for Developing Safe, Reliable, and Secure Systems in C++. <https://resources.sei.cmu.edu/downloads/secure-coding/assets/sei-cert-cpp-coding-standard-2016-v01.pdf>