Module 7 Portfolio Project Part I

Dennis Weddig

Colorado State University Global

CSC450: Programming III

Dr. Jack Li

9/28/2025 11:59pm

Module 7 Project Portfolio Part I

Repository location for Module 5 Critical Thinking assignment:

https://github.com/denniswed/csc450/tree/main/Portfolio/Module7

The below code has most notes associated with the requested assignment. I use c++20 to give me the save threads (and frankly easier to use threads so I don't have to worry about joins.) I also use std: variables to reduce risk and follow best practice, these are pointed out below with specific SEI C++ Compliant referenced sections. The thing that is not clear is around performance. Frankly, with a simple application like this, performance is a non-issue. But when dealing with threads concurrency becomes critical. The code below uses mutex's to manage the threads to prevent them from stomping each other. This is important from a stability perspective, so from the perspective of the application performing the intended purpose, using mutex's and safe threads "improve" performance. From a speed perspective, it would be faster to just let the threads run concurrently, but this does not achieve the goal. Like everything, the same applies with code, you give up some areas of performance to ensure other areas are met. In the case below, the critical functionality was to ensure the first thread finishes before the second thread. While it slows the app down, overall, it means the integrity is maintained. This is especially important when you deal with real data and the need to manage your causality of events. Even in micro-service architectures you have to deal with ensuring you are waiting for the right data and while they will not always need thread management, you do need to understand the string of events that need to happen.

Code:

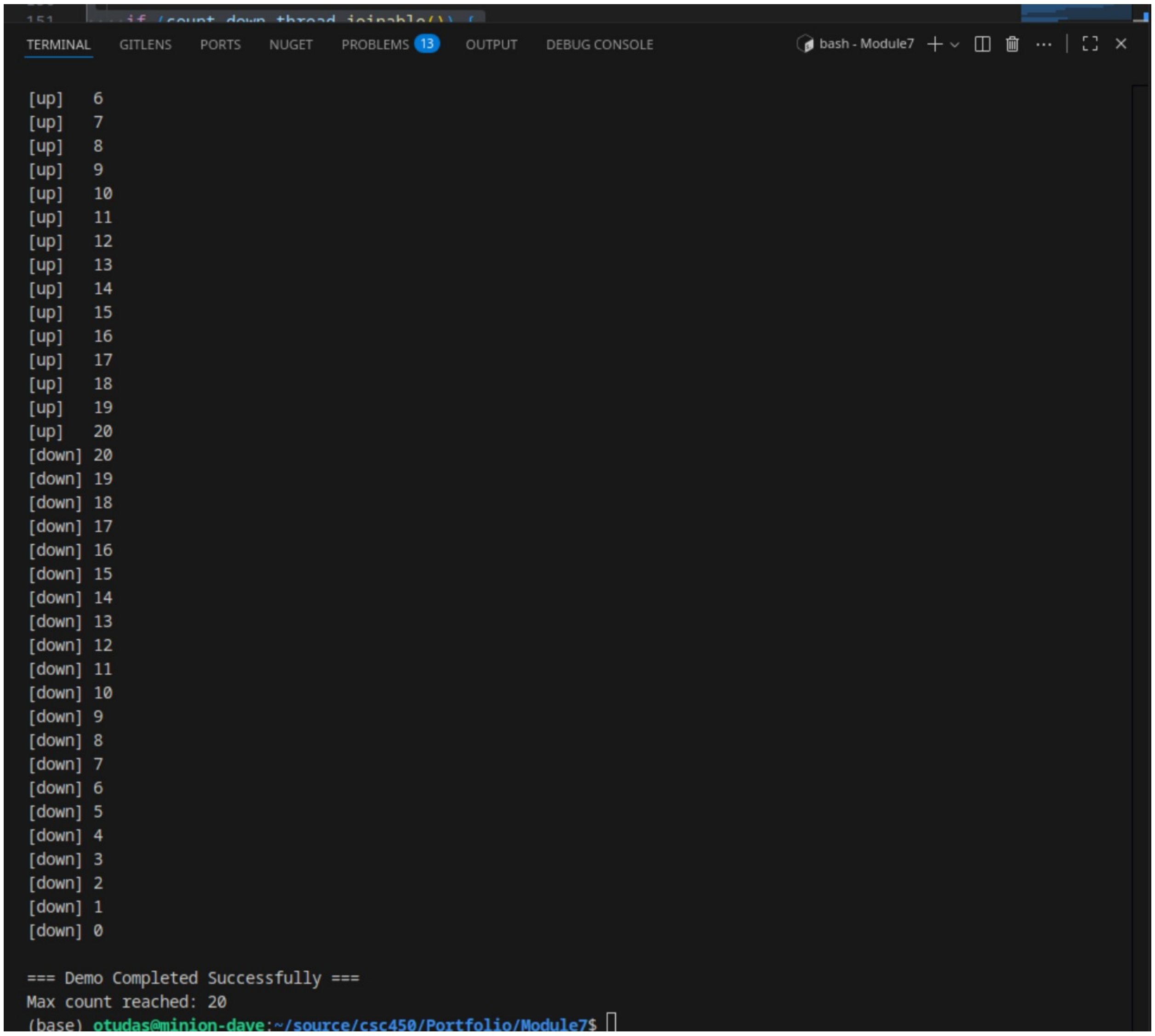
```
/**

* CSC450 Portfolio Module 7 - Personal Threading Application
* SEI CERT C++ Compliant Implementation with C++20 Features
```

```
*
* Demonstrates safe thread synchronization with modern C++ practices:
* - Thread 1: Counts up from 0 to 20
* - Thread 2: Waits for completion, then counts down from 20 to 0
*
* CERT Standards Addressed:
* - ERR50-CPP: Exception handling in thread operations
* - CON50-CPP: Proper mutex management with RAII
* - CON51-CPP: Condition variable usage with predicates
* - CON54-CPP: Spurious wakeup handling
* - DCL50-CPP: Const correctness and noexcept specifications
*/

#include <condition_variable>
#include <exception>
#include <iostream>
#include <mutex>
#include <thread>

using namespace std::chrono_literals; // C++20 convenience

namespace {

/**
* Thread-safe counter class implementing RAII and encapsulation
* Addresses OOP58-CPP (proper resource management)
*/
class ThreadSafeCounter {
private:
static constexpr int kMaxCount = 20;
static constexpr auto kPacingDelay = 5ms;

mutable std::mutex print_mutex_; // Protects output operations
mutable std::mutex state_mutex_; // Protects shared state
std::condition_variable cv_; // Coordinate between threads
bool up_complete_ = false; // Guarded by state_mutex_

public:
ThreadSafeCounter() = default;

// Non-copyable, non-movable (OOP58-CPP)
ThreadSafeCounter(const ThreadSafeCounter&) = delete;
ThreadSafeCounter& operator=(const ThreadSafeCounter&) = delete;
ThreadSafeCounter(ThreadSafeCounter&&) = delete;
ThreadSafeCounter& operator=(ThreadSafeCounter&&) = delete;
```

```
/**
* Counts up from 0 to kMaxCount and signals completion
* Thread-safe, exception-safe (ERR50-CPP)
*/
void countUp() noexcept {
try {
for (int i = 0; i <= kMaxCount; ++i) {
// RAII lock for thread-safe output (CON50-CPP)
{
std::scoped_lock output_lock(print_mutex_);
std::cout << "[up] " << i << '\n';
}

// C++20 chrono literals for cleaner code
std::this_thread::sleep_for(kPacingDelay);
}

// Signal completion (CON51-CPP)
{
std::scoped_lock state_lock(state_mutex_);
up_complete_ = true;
}
cv_.notify_one();

} catch (const std::exception& e) {
// Ensure no exceptions escape thread function (ERR50-CPP)
std::scoped_lock output_lock(print_mutex_);
std::cerr << "Error in countUp: " << e.what() << '\n';
} catch (...) {
std::scoped_lock output_lock(print_mutex_);
std::cerr << "Unknown error in countUp\n";
}
}

/**
* Waits for up completion, then counts down from kMaxCount to 0
* Proper condition variable usage with predicate (CON54-CPP)
*/
void countDown() noexcept {
try {
// Wait with predicate to handle spurious wakeups (CON54-CPP)
{
std::unique_lock state_lock(state_mutex_);
cv_.wait(state_lock, [this] { return up_complete_; });
}

for (int i = kMaxCount; i >= 0; --i) {
// RAII lock for thread-safe output (CON50-CPP)
{
std::scoped_lock output_lock(print_mutex_);
std::cout << "[down] " << i << '\n';
}

std::this_thread::sleep_for(kPacingDelay);
}

} catch (const std::exception& e) {
// Ensure no exceptions escape thread function (ERR50-CPP)
std::scoped_lock output_lock(print_mutex_);
std::cerr << "Error in countDown: " << e.what() << '\n';
} catch (...) {
std::scoped_lock output_lock(print_mutex_);
std::cerr << "Unknown error in countDown\n";
}
}

/**
* Gets the maximum count value (const correctness - DCL50-CPP)
*/
[[nodiscard]] constexpr int getMaxCount() const noexcept {
return kMaxCount;
}
};

} // anonymous namespace

/**
* Main function with proper exception handling (ERR50-CPP)
* C++20 features: designated initializers, structured bindings
*/
int main() {
try {
// Note: Deliberately NOT calling std::ios_base::sync_with_stdio(false)
// because synchronized iostreams provide stronger cross-thread safety

std::cout << "=== Thread Synchronization Demo ===\n";
std::cout << "Thread 1: Counting up to 20\n";
std::cout << "Thread 2: Will count down after Thread 1 completes\n\n";
```

```
ThreadSafeCounter counter;

// C++20: Better thread creation with member function pointers
std::jthread count_up_thread(&ThreadSafeCounter::countUp, &counter);
std::jthread count_down_thread(&ThreadSafeCounter::countDown, &counter);

// C++20 jthread automatically joins in destructor, but we can be explicit
// Verify threads are joinable before attempting join (CON52-CPP)
if (count_up_thread.joinable()) {
count_up_thread.join();
}

if (count_down_thread.joinable()) {
count_down_thread.join();
}

std::cout << "\n=== Demo Completed Successfully ===\n";
std::cout << "Max count reached: " << counter.getMaxCount() << '\n';

return EXIT_SUCCESS;

} catch (const std::system_error& e) {
std::cerr << "System error: " << e.what() << " (code: " << e.code() << ")\n";
return EXIT_FAILURE;
} catch (const std::exception& e) {
std::cerr << "Error: " << e.what() << '\n';
return EXIT_FAILURE;
} catch (...) {
std::cerr << "Unknown error occurred\n";
return EXIT_FAILURE;
}
}
```

Execution screen shots:



```
[up]   6
[up]   7
[up]   8
[up]   9
[up]   10
[up]   11
[up]   12
[up]   13
[up]   14
[up]   15
[up]   16
[up]   17
[up]   18
[up]   19
[up]   20
[down] 20
[down] 19
[down] 18
[down] 17
[down] 16
[down] 15
[down] 14
[down] 13
[down] 12
[down] 11
[down] 10
[down] 9
[down] 8
[down] 7
[down] 6
[down] 5
[down] 4
[down] 3
[down] 2
[down] 1
[down] 0

=== Demo Completed Successfully ===
Max count reached: 20
(base) otudas@minion-dave:~/source/csc450/Portfolio/Module7$
```

References