

CSE333 Exercise 12

Out: Friday, March 4

Due: Wednesday, March 9 by 11 am PST

Rating: 2 (note)

Goals

- Create a concurrent program with `pthread`s .
- Use locks to make code thread safe.

Background

A common design pattern used in concurrent programming is the Producer-Consumer pattern (https://en.wikipedia.org/wiki/Producer%E2%80%93consumer_problem). In this design pattern, there is a set of producers that generate work for a set of consumers. As the producers create work items, they place them into a buffer/queue. When a consumer is free to process a new work item, it grabs the next one from the buffer/queue.

Problem Description

In this exercise, you will take an initial sequential producer-consumer program and change it so the producers and consumer run concurrently. To make things more interesting, the producers will produce nian gao (https://en.wikipedia.org/wiki/Nian_gao) and tangyuan ([https://en.wikipedia.org/wiki/Tangyuan_\(food\)](https://en.wikipedia.org/wiki/Tangyuan_(food))) (both are popular Chinese holiday desserts) and the consumers will be responsible for eating them!

The initial program will:

1. Start a producer of nian gao,
2. Start a producer of tangyuan,
3. Start a consumer that eats all the snacks.

Since this code runs sequentially, all of the nian gao are produced first, followed by all of the tangyuan, and then all the snacks are eaten in the order in which they were produced.

Your job is to make the two producers and the consumer tasks run concurrently using `pthread`s and NOT C++11 threads. When the producers and consumer are run concurrently, the output of the program should change. For example, instead of all the snacks being made first and then all eaten, some nian gao could be made, a few could be eaten, and then a tangyuan could be made and immediately eaten. Each producer should create its snacks and add them to the queue as they are created. The consumer should remove snacks from the queue and eat them. If no snacks are present in the queue when the consumer is ready to eat another one, it should wait a short while and then check again to see if a new snack is available.

Provided Files

We have provided you with the following **four** source files, which can be downloaded from this web directory (`./ex12_files`) or with the commands:

```
bash$ wget https://courses.cs.washington.edu/courses/cse333/22wi/exercises/ex12_files/<filename>
```

- **SimpleQueue.h** — Contains the class definition for `SimpleQueue`, a linked list queue that is used as the producer-consumer buffer and is currently NOT thread safe!
- **SimpleQueue.cc** — Contains a working implementation of `SimpleQueue` that is NOT currently thread safe.
- **ex12.cc** — Contains the given producer and consumer functions plus the initial, sequential `main` code.
- **Makefile** — Provided for your convenience in compiling the executable `ex12`.

Note: You will only submit `SimpleQueue.h`, `SimpleQueue.cc`, and `ex12.cc`.

Implementation Notes

Using the `pthread` Library

You may find the following functions useful when writing your program: `pthread_create`, `pthread_exit`, `pthread_join`, `pthread_mutex_init`, `pthread_mutex_destroy`, `pthread_mutex_lock`, and `pthread_mutex_unlock`. In addition, you should understand the `mutable` keyword (<https://en.cppreference.com/w/cpp/language/cv>), because it could be necessary for making our `const` methods thread-safe.

Thread Safety

In order for the producers and consumers to run concurrently without stepping on each others' toes, you will need to make `SimpleQueue` thread safe by modifying it so it acquires a lock when it runs critical sections of code and releases it when the critical section finishes. A **critical section** of code is any section where a modifiable resource shared among threads is accessed. If critical sections are not protected by locks, bad things such as data races and data structure corruption can occur. You should use a `pthread` mutex for this part.

Concurrent Execution

Once `SimpleQueue` is thread safe, you should change `ex12.cc` so that the producers and consumer run concurrently using `pthread`s. You should NOT modify the given producer or consumer functions. Instead, you should start threads with new wrapper functions that call the desired producer/consumer function.

Data Races

For this exercise, "thread-safe" code means that it is synchronized with respect to accessing data. However, there may be inconsistencies with the usage of that data after we have read it. For example, it is possible with a thread-safe implementation to have a consumer print that they have consumed a snack before the producer has a chance to print that it has been produced. This behavior is OK for this exercise, as long as you make sure that the accessing of the shared data is thread safe.

Style Focus

Don't forget that good practices from previous exercises still apply!

Modifying Provided Code

You should change most of the starter code. You should only change what is necessary to make the program multithreaded and thread safe. This means you may not change which methods are declared `const` in `SimpleQueue`.

Error Handling & Robustness


In the case of an error, be sure to release all resources, especially when working with locks.

Submission

You will submit: `SimpleQueue.h`, `SimpleQueue.cc`, and `ex12.cc`.

Your code must:

- Compile without errors or warnings on CSE Linux machines (lab workstations, `attu`, or CSE home VM).
- Have no runtime errors, memory leaks, or memory errors (`g++` and `valgrind`).
- Be contained in the files listed above and compile with the provided Makefile.
- Have a comment at the top of your `.cc` and `.h` files with your name(s) and CSE or UW email address(es).
- Be pretty: the formatting, modularization, variable and function names, commenting, and so on should be consistent with class style guidelines. Additionally, the linter shouldn't have any complaints about your code (`cpplint.py`).
- Be robust: your code should deal with hard-to-handle/edge cases and bogus user input (if there are any) gracefully.

Submit your code on  Gradescope (<https://www.gradescope.com>). Don't forget to add your partner if you have one.

