[DRAFT] Mini-Lab: Concurrency Basics

CIS-25: Data Structures

December 1, 2025

Overview

In this lab, you'll experience two of the most dangerous bugs in concurrent programming: **race conditions** and **deadlock**. You'll observe these bugs in action, then implement fixes.

Collaboration: Submit individually, but you're welcome to help each other and share code as much as you like. Please don't copy-paste someone else's code, but feel free to work through these together and look at other people's code.

Setup

You'll need Git and CMake. Clone both repositories:

```
git clone https://github.com/CIS-25-F25-BCC/race-condition-demo
git clone https://github.com/CIS-25-F25-BCC/dining-philosophers-
    deadlock
```

Build each project:

```
cd race-condition-demo
mkdir build && cd build
cmake .. && cmake --build .
```

(Repeat for dining-philosophers-deadlock)

Problem 1: Observe the Race Condition (5 points)

1. Run the race condition demo 5 times: `./race`
2. Record the "Actual" count for each run in the table below
3. Answer the questions

Run	Expected	Actual
1	100,000,000	
2	100,000,000	
3	100,000,000	
4	100,000,000	
5	100,000,000	

Questions:

a) Why is the actual count different each time you run it?

b) The operation `counter++` looks like one step, but it's actually three. What are they?

Problem 2: Observe the Deadlock (5 points)

1. Run the dining philosophers demo: `./philosophers`
2. Watch for deadlock (program stops producing output but doesn't exit)
3. You may need to run it several times — deadlock doesn't always happen immediately
4. Press Ctrl+C to kill the program when it deadlocks

Questions:

a) What was the last output you saw before the program stopped? (Copy the last few lines)

b) In your own words, explain why deadlock occurs. What is each philosopher waiting for?

c) Describe ONE strategy that could prevent this deadlock:

Problem 3: Fix the Race Condition (5 points)

Modify `race-condition-demo/main.cpp` to fix the race condition using a mutex.

Requirements:

- Add `#include <mutex>` at the top
- Declare a global `std::mutex` (e.g., `std::mutex counterMutex;`)
- In the `incrementCounter()` function, protect `counter++` with the mutex
- Use `counterMutex.lock()` before and `counterMutex.unlock()` after

Test your fix:

1. Rebuild: `cmake --build .`
2. Run: `./race`
3. The actual count should now equal 100,000,000 every time

Paste your modified `incrementCounter()` function here:

Note: Your fixed version will be *slower* than the buggy version. Why? (Hint: think about what the threads are doing while waiting for the mutex)

Deliverables

Submit individually on Canvas:

- Problem 1: Completed table and answered questions
- Problem 2: Answered questions about deadlock
- Problem 3: Your fixed code and explanation

Grading Rubric (15 points)

- **Problem 1** (5 points): Recorded results, explained race condition
- **Problem 2** (5 points): Observed deadlock, explained why it occurs
- **Problem 3** (5 points): Correctly fixed race condition with mutex