Assignment 3: Deadlock Avoidance

CST-315 Operating Systems

Angel Velazquez, Nathan Dilla

March 3, 2024

For the first scenario, we intentionally induce a deadlock scenario sing two resources and their respective mutex locks. A deadlock occurs when two or more processes are unable to proceed because each is waiting for the other to release a resource.

For the resource1 function, the program requires two resources, in which lock1 is locked. The function performs some work related to the resource then attempts to lock lock2. Before it can lock lock2, it prints a message indicating that it is attempting to get resource 2. This leads to a deadlock scenario because while holding lock1, the function is trying to acquire lock2, but in the resource2 function, another thread is holding lock2 while trying to acquire lock1.A screen shot of a computer code

Description automatically generated

The resource2 function is virtually identical to the resource1 function.A computer screen shot of code

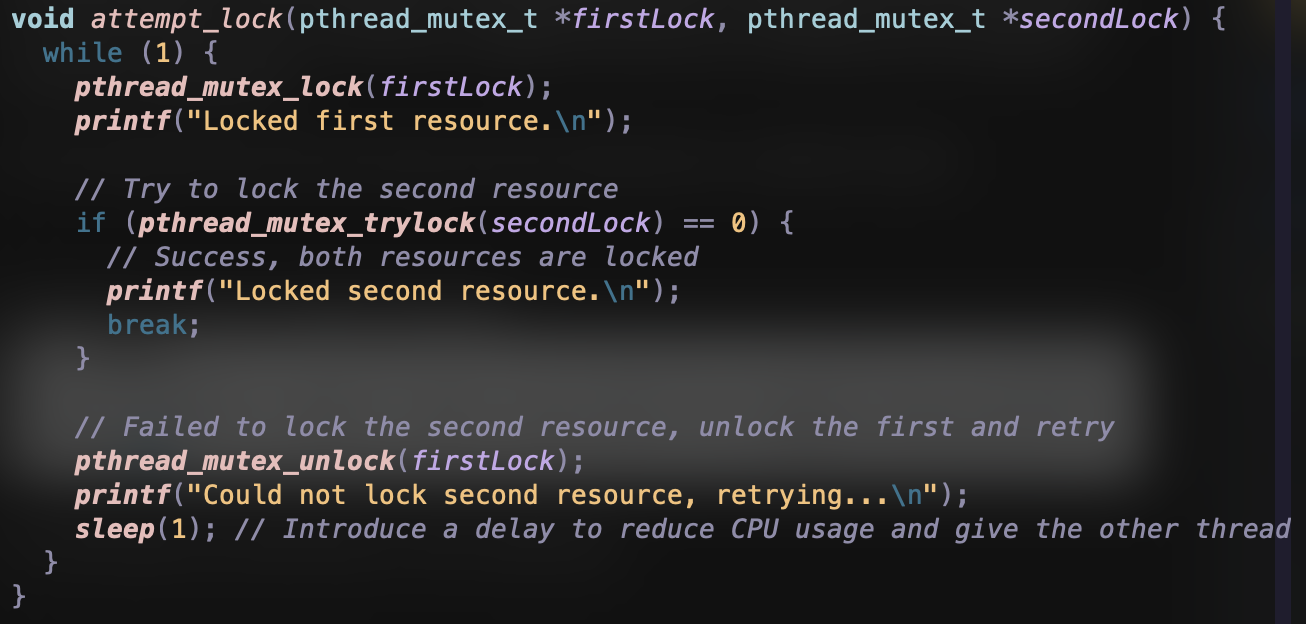
Description automatically generated

Upon execution, the program enters a deadlock and refuses to proceed.

A screen shot of a computer

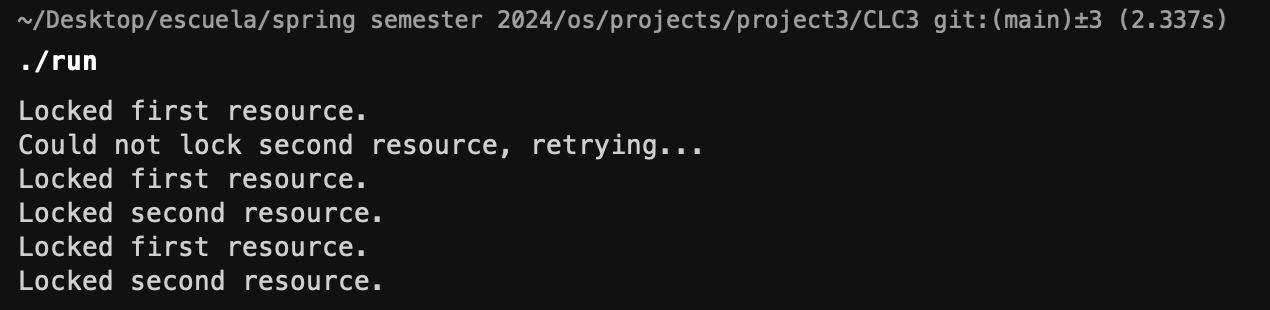
Description automatically generated

For the second scenario, we are creating two running threads that are competing for the same partition of memory. In this case, we use the try\_deadlock to prevent our 2 threads from accessing the memory simultaneously.

The screenshot below shows the logic used for utilizing the try\_deadlock method.

We use a simple if statement to check if our second lock is locked. If not, we unlock the first lock and try again at the next iteration.

To prevent the deadlock in our scenario while still demonstrating the use of both threads, we could have incorporated a strategy that involved attempting to lock both mutexes in the same iteration regardless of the thread’s operation. However, because the program uses separate functions for resource1 and resource2, it was best to utilize the try-lock mechanism. If the try-lock fails, the thread releases the first lock, waits a moment, and then attempts again. This avoids deadlock by ensuring that a thread does not hold onto a resource while waiting infinitely for another. HOWEVER, this can lead to livelock, where threads are constantly locking and unlocking resources without making progress.

The screenshot below shows both threads running, including messages that indicate their unlocked and locked states.

**GitHub Link:** [HERE](https://github.com/angel-vlzqz/Operating-Systems/tree/main/projects/CLC3)