**Assignment 3: Deadlock Avoidance**

Nolan Tuttle & Mathew Hobson

College of Engineering and Technology, Grand Canyon University

CST-315: Operating Systems Lecture and Lab

Professor Citro

March 2, 2025

**Summary**

This program implements a **thread synchronization scenario** in C where multiple threads attempt to access a shared resource, but a **watchdog thread monitors and enforces a timeout policy. A mutex** is used to control access, ensuring only one thread can modify the resource at a time. If a thread holds the resource **for more than 3 seconds**, the watchdog signals a timeout, setting a flag (active\_thread\_timed\_out). The active thread periodically checks this flag, and if set, **it releases the mutex and exits gracefully, restarting the process**. This ensures that only the problematic thread is stopped while other threads continue execution. The implementation avoids **deadlocks** by ensuring that the mutex is always released by the thread that acquired it, rather than being forcibly unlocked by the watchdog. The program runs indefinitely with both threads competing for the shared resource and deadlocks occurring at random, depending on how long each thread sleeps for (a random value between 1 and 5).

**Code Execution**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**Flowchart**

**A diagram of a work flow

AI-generated content may be incorrect.**

**Analysis**

After reviewing the activity log, the timer was somewhat efficient in solving deadlock situations. However, using a third thread as a watchdog thread uses CPU time, which is not optimal. While it is scalable, the scalability isn’t that great, if each thread must check for the signal flag telling it to stop executing, it will add more CPU time and be less efficient, which will only get worse with more threads. Deadlock prevention would be much better than this timeout solution because this requires variables to store conditions, to keep track of time, etc., taking up memory and cluttering the code. It would be a better approach would be structurally removing a condition for a deadlock, something like the mutual exclusion condition being removed so that it is impossible for a deadlock to occur. However, if simply seeking a solution for deadlock avoidance, the timer is an okay solution for small-scale applications.