Assignment 3: Deadlock Avoidance

Nolan Tuttle & Mathew Hobson

College of Engineering and Technology, Grand Canyon University

CST-315: Operating Systems Lecture and Lab

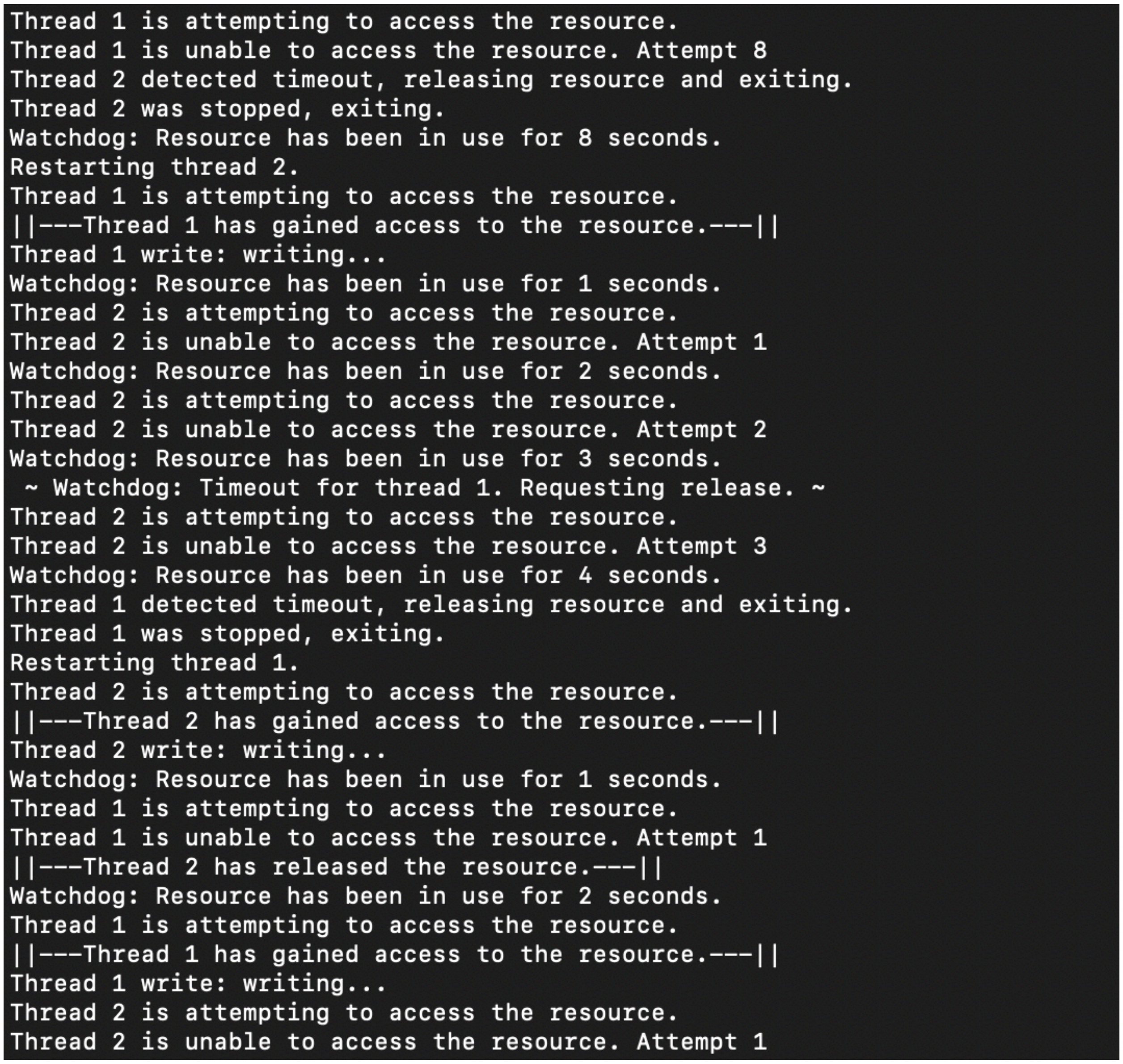
Professor Citro

March 2, 2025

Summary

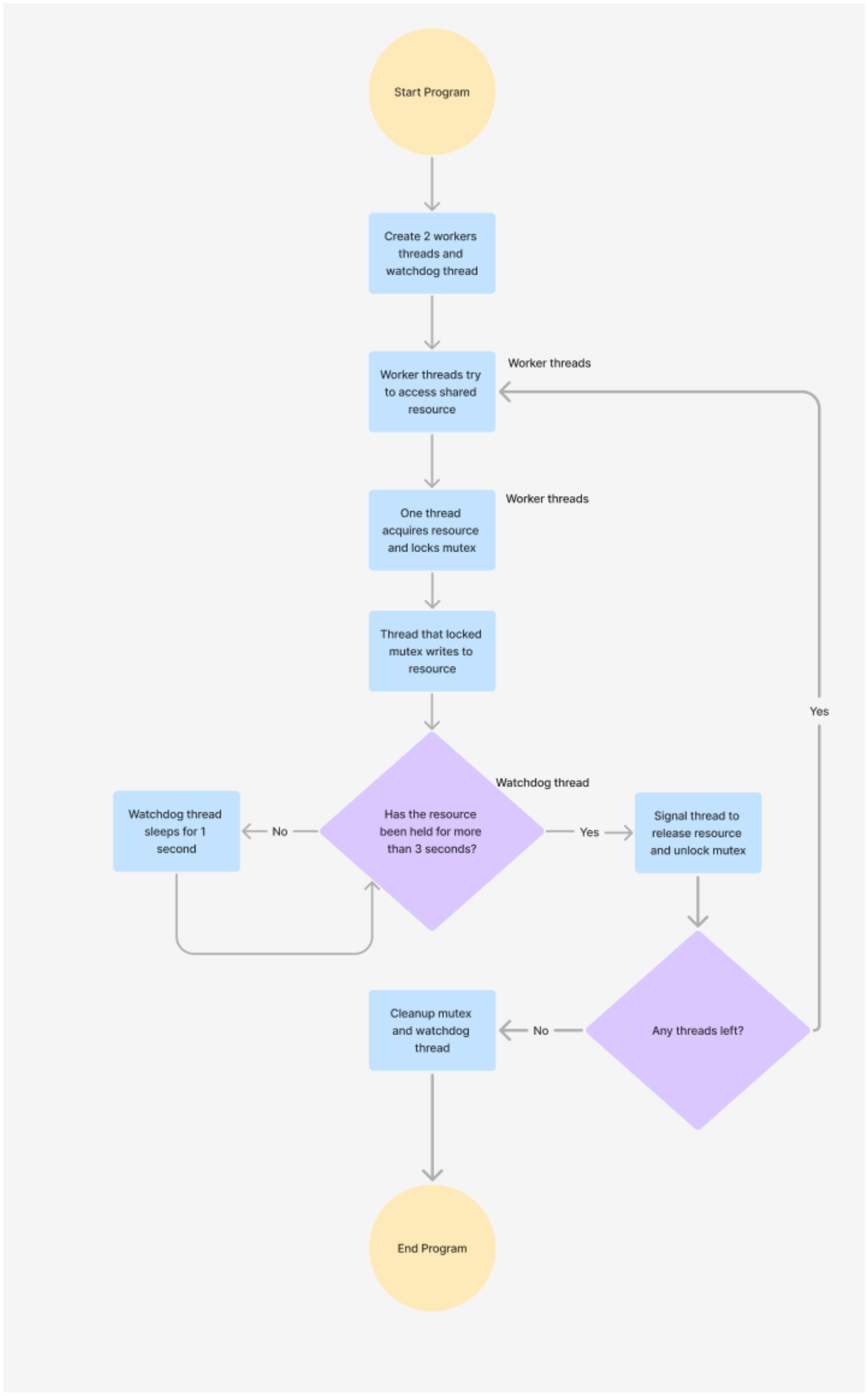
This program implements a thread synchronization scenario in C where multiple threads attempt to access a shared resource, but a watchdog thread monitors and enforces a timeout policy**. A** mutex is used to control access, ensuring only one thread can modify the resource at a time. If a thread holds 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



```
Thread 1 is attempting to access the resource.
Thread 1 is unable to access the resource. Attempt 8
Thread 2 detected timeout, releasing resource and exiting.
Thread 2 was stopped, exiting.
Watchdog: Resource has been in use for 8 seconds.
Restarting thread 2.
Thread 1 is attempting to access the resource.
||---Thread 1 has gained access to the resource.---||
Thread 1 write: writing...
Watchdog: Resource has been in use for 1 seconds.
Thread 2 is attempting to access the resource.
Thread 2 is unable to access the resource. Attempt 1
Watchdog: Resource has been in use for 2 seconds.
Thread 2 is attempting to access the resource.
Thread 2 is unable to access the resource. Attempt 2
Watchdog: Resource has been in use for 3 seconds.
 ~ Watchdog: Timeout for thread 1. Requesting release. ~
Thread 2 is attempting to access the resource.
Thread 2 is unable to access the resource. Attempt 3
Watchdog: Resource has been in use for 4 seconds.
Thread 1 detected timeout, releasing resource and exiting.
Thread 1 was stopped, exiting.
Restarting thread 1.
Thread 2 is attempting to access the resource.
||---Thread 2 has gained access to the resource.---||
Thread 2 write: writing...
Watchdog: Resource has been in use for 1 seconds.
Thread 1 is attempting to access the resource.
Thread 1 is unable to access the resource. Attempt 1
||---Thread 2 has released the resource.---||
Watchdog: Resource has been in use for 2 seconds.
Thread 1 is attempting to access the resource.
||---Thread 1 has gained access to the resource.---||
Thread 1 write: writing...
Thread 2 is attempting to access the resource.
Thread 2 is unable to access the resource. Attempt 1
```

Flowchart


Start Program
Create 2 workers threads and watchdog thread
Worker threads try to access shared resource
Worker threads
One thread acquires resource and locks mutex
Worker threads
Thread that locked mutex writes to resource
Has the resource been held for more than 3 seconds?
Watchdog thread
No
Watchdog thread sleeps for 1 second
Yes
Signal thread to release resource and unlock mutex
Any threads left?
Yes
No
Cleanup mutex and watchdog thread
End Program

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.