Dominick Girard | Assignment 7 CSCI 2503

Concurrency Programming

Avoiding Deadlocks:

- **Timeouts:** Basically, we set a timer for how long a thread can wait for a resource. If the time's up and it still hasn't got what it needs, it gives up and tries again later. This way, we don't have threads stuck waiting forever, which is a total deadlock.
- **Lock Hierarchy:** This one's about making sure threads grab resources in a specific order. It’s like forming a queue; no cutting in line or chaos ensues. By doing this, we dodge those nasty situations where each thread is holding onto something the other one needs.

Shared Resource Access:

- **Reader-Writer Locks:** When a bunch of threads just need to read a resource, but only one of them is going to change (write) it, these locks are clutch. They keep the data safe while letting multiple readers check it out at the same time.
- **Condition Variables:** These are super handy with mutexes for coordinating thread activities. Think of them as a signal system; a thread waits for the green light (or a specific condition) before it moves forward.

Possible Situations or Reasons for Necessity of Concurrency Control:

- **Real-Time Systems:** Imagine systems that need to react super fast, like in automated factories or keeping planes from crashing in air traffic control. Here, managing multiple tasks at once is critical. We can’t have delays, or things could go south real quick.
- **Database Systems:** When you've got a database handling a ton of users at the same time, keeping data consistent and in one piece is key. Concurrency control steps in to make sure everything runs smoothly without data mix-ups or crashes.

Concurrency Programming in Java

Observations and Conclusions

1. **Without Access Control:** You may observe data corruption or inconsistent reads. The write operation could interrupt read operations, leading to unexpected results.
2. **With Read-Write Locks:** Implement read-write locks to control access. The reading threads will hold the read lock, preventing the writing thread from acquiring the write lock until all read locks are released.

Why Access Control is Necessary

- **Data Integrity:** To prevent data corruption due to simultaneous read/write operations.
- **Consistency:** Ensures that threads read the latest and consistent data.
- **Prevent Race Conditions:** Avoids situations where multiple threads compete to modify/read data leading to unpredictable outcomes.