Concurrent Programming Issues: Summary

Summary of Our Discussions

- Developing and debugging concurrent programs is hard
  - Non-deterministic interleaving of instructions
- Synchronization constructs
  - Locks: mutual exclusion
  - Condition variables: conditional synchronization
  - Other primitives:
    - Semaphores
      - Binary vs. counting
      - Can be used for mutual exclusion and conditional synchronization
- How can you use these constructs effectively?
  - Develop and follow strict programming style/strategy

Programming Strategy

- Decompose the problem into objects
- Object-oriented style of programming
  - Identify shared chunk of state
  - Encapsulate shared state and synchronization variables inside objects
- Don’t manipulate shared variables or synchronization variables along with the logic associated with a thread

General Programming Strategy

- Two step process
- Threads:
  - Identify units of concurrency - these are your threads
  - Identify chunks of shared state - make each shared “thing” an object; identify methods for these objects (how will the thread access the objects?)
  - Write down the main loop for the thread
- Shared objects:
  - Identify synchronization constructs
  - Create a lock/condition variable for each constraint
  - Develop the methods - using locks and condition variables - for coordination
**Coding Style and Standards**

- Always do things the same way
- Always use locks and condition variables
- Always hold locks while operating on condition variables
- Always acquire lock at the beginning of a procedure and release it at the end
  > If it does not make sense to do this → split your procedures further
- Always use while to check conditions, not if

  ```
  while (predicate on state variable) {
    conditionVariable.wait(&lock);
  }
  ```

- (Almost) never sleep() in your code
  > Use condition variables to synchronize

**Readers/Writers: A Complete Example**

- Motivation
  > Shared databases accesses
  > Examples: bank accounts, airline seats, ...

- Two types of users
  > Readers: Never modify data
  > Writers: read and modify data

- Problem constraints
  > Using a single lock is too restrictive
  > Allow multiple readers at the same time
  > ...but only one writer at any time
  > Specific constraints
  > Readers can access database when there are no writers
  > Writers can access database when there are no readers/writers
  > Only one thread can manipulate shared variables at any time

**Readers/Writer: Solution Structure**

- Basic structure: two methods

  ```
  Database::Read() {
    Wait until no writers;
    Access database;
    check out – wake up waiting writers;
  }
  
  Database::Write() {
    Wait until no readers/writers;
    Access database;
    check out – wake up waiting readers/writers;
  }
  ```

- State variables

  ```
  AR = 0; // # of active readers
  AW = 0; // # of active writers
  WR = 0; // # of waiting readers
  WW = 0; // # of waiting writers
  Condition okToRead;
  Condition okToWrite;
  Lock lock;
  ```