Synchronization 2

Shared Variables

- Most variables in threads are local
  - The scope of the variable is limited to the thread
  - No other threads can access these variables
- Some variabless are usually shared between two or more threads
  - This is how threads communicate with each other.
    - One thread reads a value written by another thread
    - If threads are unsychronized
      - Does the reader see the value the writer writes?
      - Or, does the reader read an old, previously stored value?
  - This is the serialization problem

Other ways threads might interact?

- Concurrent reads.
  - Two (or more) threads might both read the same variable
  - Generally, not a synchronization problem
- Concurrent writes
  - Two (or more) writers
- Concurrent updates
  - Two (or more) threads perform a read followed by a write
  - Read a variable, compute new value based on old value, write new value

Concurrent Writes of a shared variable

- Variable x is shared by two writers
- What value gets printed?

Thread A

1. x = 5
2. print x

Thread B

1. x = 7

Output	Final Value (x)	Ordering (happens-before)
5	5	
7	7	
7	5	

Concurrent Updates (1)

- Is there a synchronization problem?
  - Two execution paths: A1 < B1 and B1 < A1
  - What is the result?

Thread A

1. count = count + 1

Thread B

1. count = count + 1

- In machine code, this takes two steps.

Concurrent Updates (2)

- Is there a synchronization problem?
  - Rewrite with a temp variable

Thread A

1. temp = count
2. count = temp + 1

Thread B

1. temp = count
2. count = temp + 1

- Consider a1 < b1 < b2 < a2
  - Given count = 0, what is the final value?
  - Is that what the programmer wanted? Probably not.

Atomic or Not Atomic

- Cannot always tell which instructions are performed in a single step and which can be interrupted
- Some computers have an increment instruction that is atomic
- How to write concurrent programs if we do not know which instructions are atomic and which are not?
- Conservative approach
  - Assume all updates and writes are not atomic
  - Use synchronization constraints to control concurrent access to shared variables

Thought Exercise

- Given 100 threads and count,initialized to 0, is a shared variable

```
for (int i = 0; i < 100; i++) {
    temp = count
    count = temp + 1
}
```

- What is the largest possible value of count after all threads end?
- What is the smallest possible value?

Thought Exercise

- Mutual exclusion can also be implemented with message passing
- Remember you and Bob operating a nuclear reactor?
- New.
  - You are both allowed to take a break for lunch.
  - It doesn't matter who eats first, but you and Bob must not eat at the same time
- Devise a system of message passing (phone calls) that enforces these restraints.
  - There are no clocks.
  - There is no way to predict when lunch will start or how long it will last
- What is the minimum number of messages required?

Distributed Mutual Exclusion

- In a distributed system, a shared variable (semaphore) cannot be used to implement mutual exclusion
- Message passing is the only approach
- Three approaches
  - Token-based
  - Non-token based
  - Quorum-based

Token-based Approach

- A unique token is shared by all sites
- If a site has the token, it is allowed to enter the critical section
- This approach uses a sequence number to order requests to access the critical section
  - Each request for the critical section contains a sequence number. The sequence number distinguishes old requests from new requests.
  - See Laport's Bakery Algorithm

Non-token based Approach

- A site communicates with other sites in order to determine which site should enter/access the critical section next.
  - Requires two successive rounds of message passing among sites.
- Use timestamps instead of a sequence number to order requests for the critical section

Quorum-based Approach

- Each site requests permission to execute the critical section from a subset of sites (called a quorum)
- Any two quorums contain a common site
- This common site is responsible to make sure that only one request is honored at a time