Nathan Tung

CS111 Chapter 5.2, 4.2 Outline #7

- Virtual Links using SEND/RECEIVE and a bounded buffer
  - Allows remote procedure calls between threads on same physical computer
  - Sequence Coordination with bounded buffer
    - One approach: assume the programmer makes no mistakes
    - Other approach: provide strong support for coordination correctness; loses flexibility
  - The first approach is actually better in some situations
    - Sharing a bounded buffer between two threads results in producer and consumer problem; sequence coordination is required
    - Implementation dependent on assumptions below
      - One-writer principle: one thread to send, receive, and update (each)
      - Threads run on dedicated (their own) processors
      - No overflow (long integers)
      - Read/write coherence
      - Before-or-after atomicity
  - Race Conditions
    - Error that results from a dependence on exact timing of two threads
    - Ex: Therac-25 had race conditions due to thread/operator (typing) conflicts, killed patients
  - Locks and Before-or-After Actions
    - Locks involve ACQUIRE and RELEASE, serving as a flag for coordinating shared usage of variables
    - Locks are not mechanically protective; they are only flags for convention
    - Locks help with before-or-after atomicity implementation
      - The latter is also known as isolation, isolated actions, mutual exclusions, critical sections
    - Single-acquire protocol: only one thread can acquire a given lock at a given time
  - Deadlock
    - Condition in which a group of threads is waiting for some other thread to make progress
    - Represented using a wait-for graph (nodes are threads/locks, edges are acquires)
    - Might even occur with a single lock due to, say, waiting for buffer space, etc.
    - Livelock: condition in which thread is repeating some operations but not the entire sequence
  - ACQUIRE and RELEASE can be reduced by bootstrapping, an inductive proof
    - RSM (read and set memory) performs statements as before-or-after actions
  - Coordination between Synchronous Islands with Asynchronous Connections
    - Arbiter failure: not reaching a decision before the clock ticks
    - Avoid by
      - Synchronizing clocks of two components
      - Design arbiters with multiple stages
      - Stop clock of synchronous component
      - Make all components asynchronous
- Communication between client and service
  - Remote procedure call (RPC): stylized form of client/service interaction
    - Each request is followed by response
    - System goal: make RPC look like ordinary procedure call
    - Also, it is sometimes desirable to be able to send messages to offline recipients
  - Stubs: procedures that hide marshaling/communication details from caller/callee
  - RPCs differ from regular procedure calls
    - RPCs can reduce fate sharing between caller and callee by exposing failures of callee to caller
    - RPCs introduce new failures
    - RPCs take more time (stub, (un)marshaling arguments, sending request over network, etc.)
    - Some programming languages don't combine well with RPC
  - No response case handled by implementions:
    - At-least-once RPC
      - Stub resends requests until service responds
      - Idempotent: for some operations, repeating same request has no effect

        - At-most-once RPC
            - No response with specific time means caller receives an error
            - Important for operations with side effects
        - Exactly-once RPC
            - Ideal, but independence between client/service makes this principle impossible to guarantee
            - Return error status, sends a separate RPC request to inquire about status, etc.
            - Requires both client/service stubs to track RPC requests/responses
    - Communicating through intermediary
        - Use buffered communication to allow sender/receive to be not present at the same time
            - Sender/receiver can push or pull messages
            - Decouple modules with indirection: intermediary determines message recipient
            - Choice of when/where to duplicate messages
        - Publish/subscribe is a general communication style
            - Sender is publisher, notifies event service of new message
            - Recipients subscribe to event services