Nathan Tung

CS111 Chapter 9.1 Outline #8

- All-or-nothing atomicity vs. before-or-after atomicity
  - Atomicity Example
    - Power failure while online purchasing
      - Did your order go through? Was your card charged but the item not set to ship?
    - Two people online purchasing the last item
      - Who gets the out of stock message? Who gets the item?
  - Atomicity: performing of sequence of steps (actions) appearing as a single, indivisible step (atomic action or transaction)
    - Atomic action provides all-or-nothing atomicity (even if fault occurs halfway through)
    - Sweeping simplification
      - Simplifies design, understanding, and correctness
      - Allows failure recovery of coordinating concurrent activities
  - Ways to achieve atomicity: version histories, logging, locking protocols
- Atomicity required in computer design
  - Needed to manage database, develop hardware architecture, specify OS interface/software engineering, etc.
  - All-or-nothing atomicity…in database
    - Making sure put/get in a bank account transfer doesn't get broken up
  - Interrupt interface
    - Instruction that caused exception must tell handler either it completed or it did nothing
      - All-or-nothing report
  - Layered application
    - Carefully implemented Java interpreter will hide machine language from programmer
    - Error containment at different levels; allows systematic responses to faults
    - Some layers complete; others back out; but all actions are all-or-nothing
  - Careful design and specification is needed for all-or-nothing atomicity (unintelligible error messages are a bad sign)
- Other actions with/without all-or-nothing atomicity
  - Multilevel memory management in computer architecture
  - Supervisor call (SVC)
    - Nothing option: blocking READ
    - All option: non-blocking READ
    - Non-atomic blocking read design
- Coordinating concurrent threads
  - Actions from invoker view appeared either completely before or completely after one another
  - Different from sequence coordination
    - Programmer doesn't know actions that might touch shared variables
- Correctness and Serialization
  - Correctness between coordination among concurrent actions
    - Results are is if they could have been obtained by purely serial application of those actions
      - Old system state correct from application point of view
      - Action by itself transforms correctly from old to new state
      - If the above two are true, the new state must also be correct
  - Before-or-after atomicity effectively serializes the actions, guaranteeing correct coordination
    - Serializable: some serial order of concurrent transactions would lead to same ending state
- Summary
  - Atomic action: there is no way for higher layer to discover internal structure of its implementation
  - From procedure's POV, the action also either completes or does nothing; helps failure recovery
  - From concurrent thread POV, actions occurs completely before or after other concurrent atomic actions
  - More benevolent side effects: audit logs which record failure details; performance optimization occurs when atomic action fails and aborts, but its actions might have optimized the next successful action