Nathan Tung

CS111 Chapter 2.1 Outline #3

- Overview
  - Three main, fundamental abstractions
    - Memory (number of data items to remember)
    - Interpreter (number of steps the interpreter must execute)
    - Communication Link (number of messages to communicate)
  - These are used to organize physical hardware structures
    - They supply functions of recall, processing, and communication
    - Widely useful with understandably-simple interface semantics
  - System designers rearrange/repackage these abstractions, layering them into more customized ones
  - Reference – primary method by which abstract components in the system interact
    - Usual method for components to connect is by "name," which appear in their interfaces
    - Memory stores/retrieves by name; interpreter manipulates by name; names identify communication links
- **Memory** (storage)
  - System component that remembers data values for computation
  - Simple abstract model for all memory devices
    - WRITE(name, value)
      - Value to be remembered; it can be recalled by its name
    - value ←READ(name)
      - Memory device remembers value associated with given name
  - Volatile (consumes energy to retain energy; interrupted power means information loss)
    - When connected to a battery or uninterruptable power supply, it becomes durable (able to remember for some period, or "durability")
  - Non-volatile or stable storage (always retains content; power means READ operations can be run)
  - Both types of memory face decay, so every such device has a durability
  - Memory devices don't usually name, READ, or WRITE values of arbitrary size
    - Operates contiguous arrays of (fixed) bits, or bytes (higher-layer uses record, segments, or files)
  - Memory/storage cell: unit of physical layer memory written or read
- Read/Write Coherence & Before-Or-After Atomicity
  - Read/Write Coherence: result of READing a named cell is the same as the most recent WRITE to that cell
  - Before-Or-After Atomicity: every READ/WRITE result occurs distinctly before/after other R/W operations
- Threats against Read/Write Coherence and Atomicity
  - Concurrency: concurrent READ/WRITEs on same cell needs arbitration so one occurs completely before
  - Remote storage: physical distance and lag may make unclear which operation was most recent
  - Performance enhancements: processors might reorder operations for optimization, thus destroying concurrency; programmers can force code to run concurrently (SYNCHRONIZED; memory barrier)
  - Cell size incommensurate with value size: large values may take up multiple cells, and a concurrent READ/WRITE may cause "write tearing," or only some cells of that value are updated; small values may share a single cell, and concurrent operations might overwrite the other; atomic coordination is needed
  - Tradeoff between efficiency and R/W Coherence and Atomicity in a system design
- Memory latency (access time)
  - Time taken for a READ/WRITE to complete
  - Magnetic disk memory latency depends on the device's mechanical state at request instance
    - If the disk rotates past a second nearby sector that the user wants to access, there may be a thousand-time delay as we wait for the next rotation under the read head
    - Max transfer rate to a disk is much larger than that of random sectors
  - Random access memory (RAM)
    - Latency of randomly-chosen memory cells is about the same as those chosen in the best order
    - Behavior unlike other mechanical memory, like optical disks or magnetic tapes/disks
    - For non-random memory, it is more worthwhile to READ/WRITE huge blocks
- Memory names/addresses
  - Memory cells are generally named by the geometric coordinates of their physical storage location

  - Geometric coordinates are sometimes mapped in consecutive integers called "addresses," which form the address space of the memory device
  - Such a memory system is called location-addressed memory
  - Associative memory: memory system that accepts unconstrained names (unlike consecutive integers)
  - Cache: device that remembers the result of an expensive computation in order to reuse it
    - Often associative hardware/software
- Exploiting memory abstraction (RAID)
  - Redundant Array of Independent/Inexpensive Disks
  - An example of storage abstraction and modularity with two goals
    - Improved performance by reading/writing disks concurrently
    - Improved durability by writing information on more than one disk
    - Various configurations, such as allowing concurrent read/write, error correcting, or copying data
- **Interpreter**
  - Active elements in a computer system for performing computational actions
  - Abstraction of three components
    - Instruction reference: tells interpreter where to find next instruction
    - Repertoire: set of ready-to-perform actions by interpreter when instruction is found at location
    - Environment reference: tells interpreter where to find environment (or state on which to act)
  - Interpreter normally sequences through the program, finds environment by environment reference, takes the program instruction from that, then performs the action by changing some data; repeat
  - Interrupts (handler) might provide the interpreter with code rather than the program's instruction code
  - Multiple interpreters are often asynchronous (run on uncoordinated clocks) and run at different rates
    - Contributes to coherence/atomicity problems in memory
- Processor
    - Implementation of an interpreter; instruction reference is a program counter in the processor's fast memory processor; may be wired directly to memory; addresses in the program counter are names in that memory's address space
    - The repertoire includes instructions that express computations or move data
    - A stack is provided for implementing procedure calls
      - Caller pushes callee (arguments) on the stack; when the callee returns, stack pops back
      - Stack pointer register holds memory address of top of stack for efficiency
    - Interrupts may occur due to an instruction that cannot be implemented or an external signal that needs attention
      - Exception handler may take over in the former case
      - In the latter case, interrupt handler may do some work, then return control
- Interpreter Layers
  - Interpreters are organized in layers (a full application might have 4 or 5 layers)
  - Lowest layer is hardware with primitive repertoire; successive layers have more functionality
  - Example: hardware → interpreter → program → (user)
  - Goal: ensure designers that the layer under another layer either completes instruction or does nothing
  - Assume interpreters are atomic
- **Communication Link**
  - Provides a means for information to move between physically-separated components
  - SEND(link_name, outgoing_message_buffer)
    - An array of bits (a message) is transmitted across link_name
  - RECEIVE(link_name, incoming_message_buffer)
    - Some buffer is specified to hold the incoming message
    - Higher interfaces can RECEIVE from lower ones (lower ones can "upcall," or deliver)
  - The link_name argument is a possible communication link; some links are multiply-attached networks of links with another recipient-choosing method
- Differences from READ/WRITE make the semantics and implementation rather different
  - Parameters make SEND/RECEIVE times unpredictable; if asynchronous, time is unknown in advance
  - Environment threatens integrity of data transfer; in some cases, the message might fail to deliver