CSE 461 - Module 3: Software Structure

General Issue: Implementing Concurrency

- Correctness vs. performance
  - Abstract correctness
    - App provides correct results, ignoring all aspects of time
  - Behavioral correctness
    - Using app “feels” right
  - Performance
    - How much (hardware) resource is required to provide behavioral correctness
- Implementation ease / difficulties
  - Can I understand the code I've written?

Example App: Web Server

- Operation
  - Browser connection occurs
    - Server parses request and determines what file to return
    - Server reads the file
      - Server processes the file (server side includes)
      - Server reads included file
      - Repeat
    - Server writes result to socket
  - Repeat

Computational Concurrency Alternatives

- Single threaded
- Multi-threaded
  - (usually not fork-join parallelism)
- Thread pool / work queue

I/O Alternatives

http://www.ibm.com/developerworks/library/l-async/

	Blocking	**Non-Blocking**

Synchronous	Read / write	`flags |= O_NONBLOCK;`
		`fcntl(fd, F_SETFL, flags);`
		
		`Read / write`

Asynchronous	Select / poll Non-blocking read/ write	aio interface (start op; completion event)

- Providing concurrency with the **synchronous, blocking model requires threads**
  - A thread per what?
  - Pro's:
    - Can often be most like single-threaded code (so is simple)
    - Threads naturally encode state when performing a sequence of operations
  - Con's:
    - Can be slow
    - Potential for leaks – threads are a resource
    - Potential for race conditions
      - Use thread-safe data structures / atomic data types
    - Potential for (local) deadlock
- **synchronous, non-blocking**
  - Pro's:
    - Probably don't need dynamic thread creation
  - Con's:
    - What to do while waiting for IO
      - Spinning
    - Often leads to complicated code
- **asynchronous, blocking**
  - CSE 333 Non-blocking IO / Select lecture slides
  - Pro's:
    - Probably don't need dynamic thread creation
    - `select` provides solution to spinning issue
  - Con's:

        - Can be slow if there are 1000's of connections
        - Possible race conditions (between `select` call and acting on its return values)

Summary

- Choosing an appropriate code structure can be complicated
    - http://www.kegel.com/c10k.html
- Our priorities in projects:
    1. abstract functional correctness
    2. ease of development / maintenance
    3. efficiency
    4. scalability