CSE 550: Introduction to Computer Systems Research

Arvind Krishnamurthy
Course Information

- Instructor: Arvind Krishnamurthy
  - Interests: distributed systems, networks, operating systems, security
  - Email, office hours on the website
  - Also fine to just drop in!

- TA: Niel Lebeck
Course Basics

- Quals course that covers foundational systems topics from:
  - Operating Systems, Networks, Distributed Systems, Databases
- No prerequisite
- Gateway course to CSE 551, 552, and 561 or a terminal course for students desiring breadth
What is a computer system?

• Our focus is on software systems

• Software system achieves a specific external behavior (e.g., deliver videos, online social network, email)
  • Might operate only under certain assumptions

• Comprises of many components
  • Components interact and cooperate to provide overall behavior
  • They typically have specified interfaces
Thought Exercise

• Let us say that you want to build a Youtube-like service
  • What are the key components in its design?
  • What are the key issues/tradeoffs?
Course Topics

- Concurrency
- Web Services
- Local Transactions
- Distributed Transactions
- Distributed clocks
- Consensus/RSM
- Virtualization
- Software Virtual Memory
- File systems
- Large storage systems
- Consistent storage
- DHTs
- Big data
- Networking (cong. control)
- Networking (routing)
- Experiences
Course Format

Three components:

- Reading papers and blog posts on papers
  - Respond to questions or email us a summary
- Programming assignments in teams of two
  - First assignment out, due in two weeks
- Course project
Key Goals in Systems

- Correctness
- Availability/reliability
- Security
- Performance
What makes achieving these goals hard?

- **System complexity:**
  - Large # of components
  - Large # of connections
  - Irregular interactions, irregular resource needs
  - Imprecise description, many required to design/maintain

- **Technology rarely the limit!**
  - Limit is usually the complexity, ability to abstract, reason, etc.
Sources of Problems

• Emergent behavior or surprises
• Propagation of effects
• Unexpected scaling
Example

- Amazon EC2 outage from few years back

**Background on EC2:**
- Multiple regions; multiple “availability zones” within each region
- Each availability zone provides the Elastic Block Store (EBS)
  - EBS volumes are mountable on EC2 nodes
  - Replicated to deal with faults
  - EBS nodes use a “peer-to-peer” protocol to detect faults and replicate; blocks while trying to replicate
  - EBS nodes connected by a backup lower capacity network for providing reliable control
- Control plane keeps track of volume locations; replicated/shared across the entire region
Outage

- Configuration change to upgrade a router
  - Normally shift traffic off to a full-capacity redundant router
  - Instead, mistakenly assigned to the backup router which overloaded
- EBS nodes weren’t able to contact each other, so declared failure and tried to provision extra copies
  - Exhausted space. Created a “re-mirroring” storm.
- Created a huge load on the control plane
  - Overload caused control plane to not handle operations from other availability zones
  - Operators recognized the problem and disabled “re-mirroring” operations
- Caused further problems! No aggressive back-off, a race condition in EBS nodes in closing connections -- which caused them to fail resulting in further re-mirroring. Operators finally disconnected the availability zone.
• What are the take-aways from this incident?
Unix Time Sharing System

• Classic system and paper: described almost entirely in 10 pages

• Key idea: elegant combination of a few concepts that fit together well

• Third system for time sharing:
  • First system was CTSS an unqualified success
  • Followed by Multics, which suffered from the second system effect
Unix

- Designed by Ritchie and Thompson
- Platform: PDP-11 computer; operational in 1971
- Written in C (instead of assembly -- 33% overhead)
- 2 man-years to write
- size < 50 KB
- Defined an ecosystem of related tools
  - Written collaboratively
  - Developers used/built the system for their own work
Unix Components

- File systems (ordinary files and device I/O)
- Process management
- Shell

Question: is there anything missing from the above list?
File System

- “Important job of Unix is to provide a file system”
- Three types of files:
  - Ordinary files: sequence of bytes (unstructured)
  - Directories (protected ordinary files)
  - Special files (I/O)
- Question: should they have supported other types of files? If so, what?
Directories

- Map: names of files to file location on disk
- Hierarchical
- Manipulated by programs that have appropriate permissions
- Linking: file does not exist within a particular directory
  - Directory entry merely contains a pointer to the file descriptor that describes the file
Special Files

- **Uniform I/O model**
  - open, close, read, write, seek

- **Uniform naming and protection model**
  - user-world, RWX bits
  - set-user-id bit
  - super user is just special user id
Removable File System

- Tree structured
- "mount"-ed on an ordinary file
  - Associate a special device file with an ordinary file inside the tree structure
File System Implementation

- Table of i-nodes
- Path name scanning
- Mount table
- Buffered data
- Write-behind
Processes

- Text, data, and stack segments
  - Text is shared, the rest are process-specific
- Process swapping
- fork, exec: create new processes from same or different images
- Pipes for communicating between processes
- wait, exit: synchronization primitives
Shell

- Invoke programs: “cmd arg1 ... argn”
- Performs stdio and I/O redirection
- Filters & pipes
- Multi-tasking from a single shell
- Shell is just a program!
Questions

- What are the key design principles employed in Unix?
- What has changed and what hasn’t?
- What would you do differently for different settings (e.g., handheld devices)?
- How would you evaluate this paper now?