
Lecture 1: Welcome to CSC469

Advanced Operating Systems

Instructor: Angela Demke Brown

Email: demke@cs.toronto.edu

Website: <http://www.cs.toronto.edu/~demke/469F.06/>

Plan for today

- Overview of CSC 469
 - How it'll work
 - What I expect from you
- What makes software systems tough and interesting
 - Reality
 - Complexity
- Goals and Topics
- What's next...

Overview of 469 (Fall 2006)

- Check the web page for updates and news frequently
 - <http://www.cs.toronto.edu/~demke/469F.06/>
- Components
 - Regular lectures (by me) and discussion (by you)
 - Tutorials (concrete examples, assignment help, Q&A)
 - Four assignments
 - Two term tests
- Other stuff
 - Readings from the research literature will be assigned
 - No required text, but some books on reserve in library
 - Prereqs

How to read a research paper

- Consider the source (don't dismiss, but do consider)
 - Who wrote it -- are they experts or unknowns?
 - Where was it published -- top journal or personal web page?
 - Other aspects: sponsor, review process, structure, tone, etc.
- Dig for the point
 - Read the abstract, intro, conclusion and related work (and bib)
 - Flip (semi-quickly) thru the paper, looking at headings, figures and data
 - Consider how much time you really want to devote to the guts
 - What is the hypothesis, how do they try to prove it, and do they succeed?
- Computer Systems papers
 - Often describe entire systems without a clear point or hypothesis

Assignments

Goal is to explore different operating systems and the impact of design choices...

- Assignment 0 - Benchmarking: System structure and performance (due Oct. 5)
- Assignment 1 - Using advanced synchronization (due Oct. 19)
- Assignment 2 - Code comparison: multiprocessor OS support (due Nov. 9)
- Assignment 3 - TBD (due Nov. 30)

Prereqs and Refresher

- Prereq: CSC 369 or equivalent
 - you should have a solid command of this material
 - if you don't, you will struggle
 - worse, you will not benefit nearly as much as you should
- Handout #1 & #2: helping you refresh
 - a bunch of questions from ACM Self Assessment Procs
 - the point is to swap in your OS knowledge
 - use your OS book from 369
 - discuss the problems and topics with your peers
 - **now** is the time to refresh your memory!
 - so, we will be discussing these questions in tutorial on Thursday

Making the grade in 469

- Breakdown
 - 4 Assignments (10%, 10%, 15%, 15%)
 - First one solo, rest you can pair up
 - 2 term tests (Oct. 26 - 20%, Dec. 7 - 25%)
 - 5% class discussions
- You need to be here (and in tutorial) to participate in discussions and get the most out of the class!

What makes software systems tough and interesting

- Reality
 - simplifying assumptions often don't hold up
 - people are rarely rational, arrivals are rarely Gaussian, environments are rarely clean, failures are rarely independent, etc...
 - poorly done systems can be incredibly expensive
 - billions of dollars and even life & death
- Rapid changes in technology and applications
 - technology advances change the rules
 - new applications change the requirements
- Most generally: **Complexity**
 - coping with complexity is what almost all of it boils down to

Real examples of disasters (from J. Saltzer)

- Tax system modernization
 - U.S. Internal Revenue Service
 - tried to replace 27 aging systems
 - Started 1989, scrapped 1997, spent: \$4B
 - Causes: complexity
 - all-or-nothing massive upgrade
- Advanced automation system
 - U.S. Federal Aviation Administration
 - tried to replace 1972 Air Route Traffic Control System
 - Started 1982, scrapped 1994, spent: \$6B
 - Causes: complexity
 - changing specifications, grandiose expectations, congressional meddling

More examples of disasters (from J. Saltzer)

- Ambulance dispatching
 - London ambulance service
 - automate dispatching and routing
 - Started 1991, scrapped 1992, cost: 20 lives lost (and \$2.5M)
 - shut down after 2 days of operation
 - Causes: complexity and poor management
 - unrealistic schedule (5 months), overambitious objectives, unidentifiable project manager, low bidder had no experience, no testing/overlap with old system, users not consulted during design
- Automated DMV
 - California department of motor vehicles
 - Started 1987, scrapped 1994, spent: \$44M
 - Causes: complexity and blame shedding
 - underestimated cost by 3X, slower than 1965 system, governor fired whistleblower, DMV blames Tandem, Tandem blames DMV

Rapid pace of our field

- Technology is a major driver
 - Technology eliminates some problems and creates new ones (and enables new applications) over time
 - Incommensurate scaling makes things interesting
 - This means that one has to be on top of technology characteristics and trends
- New application requirements are another major driver
 - Changes the rules (assumptions), often forcing redesign
 - Example: video conferencing vs. best-effort networking
 - Example: mobile computing vs. file system caching
- Systems are complicated and consist of many parts
 - To do top-quality work, you must know about them all!
 - ... and their interactions too.

Problems in complex systems

- Propagation of effects
 - a “small, localized change” often has far-reaching effects
 - example: 13-inch tire to 15-inch tire to improve tire lifetime
 - consequences: wheel wells must be enlarged, spare tire space must be enlarged, back seat must be moved forward to accommodate spare
 - “there are no small changes in a large system”
- Surprises
 - an unexpected consequence of a change
 - example: 15-inch tires may weigh 21 pounds
 - consequence: exceed limit of current land shipping mechanism
- Incommensurate scaling
 - as a system scales up or down in size or speed, not all parts of it follow the same scaling rules
 - example: a mouse the size of an elephant would collapse

How does this translate to software systems?

- The software systems we're concerned with suffer from all of these problems
- We'd like to have a constructive theory to apply
 - e.g., like linear control systems, thermodynamic systems, ...
- Unfortunately, we don't
 - note that this would be a worthy career contribution
- Where does that leave us?
 - Case studies
 - Lessons from study of other complex systems
 - major difference is the unprecedented rate of change

Let's try to define complexity, as a start

- Webster: "the state of being complex"
 - complex => "difficult to understand"
- Relative term, not lending itself to quantification
- Symptoms of complexity
 - large number of elements
 - large number of interconnections
 - irregularity (lots of exceptions, neither regular nor repetitive)
 - lack of a methodical description
 - like previous one, but highlights difficulty of understanding
 - minimum team size
 - combines all of above into "how many people to collectively get it"

Some sources of system complexity

- Large number of objectives
 - example: new goals, requirements, or performance targets
- Need for high utilization of limited resources
 - example: single-track railroad line
- Generality
 - example: separately steerable front wheels
 - generally, generality increases complexity
 - frequently, it does so without real purpose

Techniques for coping with complexity

- Modularity
 - divide-and-conquer can often reduce growth (as function of size) from square to linear
- Abstraction
 - separation of interface from internals
 - or specification from implementation
- Hierarchy
 - builds on modularity by recursively grouping module sets
 - most surviving complex systems use it: army, company, etc

Some add'l practical techniques

- Control novelty
 - sources of excessive novelty
 - second-system effect, better technology, marketing pressure...
 - some novelty is necessary; the hard part is figuring out when to say NO
- Install feedback
 - design for iteration, iterate the design
 - something simple working first; one new problem at a time
- Find bad ideas fast
 - understand the design loop and examine the initial requirements
 - try ideas out, but don't hesitate to scrap them
- Conceptual integrity
 - one mind controls design
 - good aesthetics yield better systems
 - also makes them much easier to debug and maintain

What was the point of that?

- Well, what do operating systems do?
 - Provide abstractions of system resources
 - processes, files, sockets, malloc, etc...
 - Isolate application writers from details and each other
- Also, tend to be highly complex
 - many objectives
 - performance, reliability, ease of use, security, maintainability, ...
 - desire for high utilization of resources
 - generality: support all applications well
- Also, the problem keeps changing
 - technology advances and new applications
- ... and combining multiple systems (the soul of distributed systems) complicates everything further

Major goals of 469

- Understand how OS design changes in response to
 - key technological advances (e.g., CPU vs. disk)
 - new application requirements (e.g., mobility, QoS)
 - advanced system objectives (e.g., fault tolerance, security)
- Understand key aspects of distributed systems
 - getting systems to talk to each other
 - marshalling separate resources to achieve common goals
 - figuring out where to perform particular functions
- Approach: case studies and existing experience

Major 469 Topics

- Operating system structure and internals
- Performance evaluation and benchmarking
- Communication models
- Concurrency, distributed event ordering, multi-party consensus
- Multiprocessor operating system issues
- Advanced virtual memory and storage systems
- Distributed systems
- Security
- Future environs

What's next ...

- Page in your OS knowledge from 369!
- Study options for operating system structure