The slides are the product of many rounds of teaching CS 4410 by Professors Sirer, Bracy, George, and Van Renesse.

### CS 4410/4411 Operating Systems

**Professor Bracy & Van Renesse**

Fall 2015

Cornell University

---

**About Prof. van Renesse**

**Research Interests:** scalability, fault tolerance, security, OSs, and more
- Has worked on Amoeba and Plan 9 operating systems
- Best known research artifacts: Chain Replication, Scuttlebutt (used in Cassandra, etc.)

**Other Interests:** Playing music (trad. jazz), swing dancing, unicycling

---

**About Prof. Bracy**

**Professional Interests:**
- **Teaching:** intro to programming, digital design, computer architecture, system software,
- **Research:** computer architecture, instruction fusion, hardware support for thread synchronization

**New to Cornell**
- Previously @ Washington University in St. Louis
- Prior to that: research @ Intel Labs
- Also teaching 3410 this semester (and will be teaching both again next semester)

**Other Interests:** at the moment ...unpacking

---

**About You...**

**Demographics, etc.**
What’s This Course About?

• *Ostensibly:* operating systems
  - architecting complex software
  - identifying needs and priorities
  - separating concerns
  - implementing artifacts with desired properties

• *Actually:* software design principles
  - OSes illustrate important organizational principles and design patterns
  - Could also teach, say, building browsers, as an excuse for teaching these principles

What Kind of a Course Is This?

• Constructive, top-down
  - Start from first principles and re-derive the design of every component of a complex system

• Deconstructive, bottom-up
  - Dissect existing systems, learn what tradeoffs they make, what patterns they use

• *Warning:* lots of new jargon, terms, etc. to be learned

Course Objective

• Establish a foundation for building complex programs
  - Architect systems
  - Identify desirable properties
  - Build large systems
  - Debug complex systems

• Understand all layers of the *software stack* between hardware and applications

What is an Operating System?

• An OS provides a virtual execution environment on top of hardware that is more convenient than the raw hardware interface
  - “All of the code you did not write”
  - Simpler
  - More reliable
  - More secure
  - More portable
  - More efficient

Applications

Operating System

Hardware
OS: a collection of abstractions

- Processes (abstract CPUs)
- Files (abstract disks)
- Network Endpoints (abstract NICs)
- Windows (abstract screens)
- ...

you can think of these as objects with state and methods

Abstractions are designed

- Not a consequence of logical reasoning
- A good abstraction
  - is portable and hides implementation details
  - has an intuitive and easy-to-use interface
  - can be instantiated many times
  - is efficient and reasonably easy to implement

What do OS abstractions do?

- Manage physical resources
- Provide virtual resources
- Implement mechanisms and enforce policies to arbitrate access to resources
- Mediate the interaction of mutually distrusting applications
- Provide an extensible, general-purpose platform for a variety of applications

What Resources?

- Physical Resources
  - CPU, memory, disks, screen, keyboard, mouse, networks, I/O devices, ...
- Virtual Resources
  - Files, directories, sockets, windows, names, ...
Issues in OS Design

- **Structure**: how is an OS organized?
- **Concurrency**: how are parallel activities created and controlled?
- **Sharing**: how are resources shared?
- **Naming**: how are resources named by users?
- **Protection**: how are distrusting parties protected from each other?
- **Security**: how to authenticate, authorize and ensure privacy?
- **Performance**: how to make fast?

More Issues

- **Reliability**: how do we deal with failures?
- **Portability**: how to write once, run anywhere?
- **Extensibility**: how do we add new features?
- **Communication**: how do we exchange information?
- **Scale**: what happens as demands increase?
- **Persistence**: how do we make information outlast the processes that created it?
- **Accounting**: who pays the bills and how do we control resource usage?

Why Learn Operating Systems?

**Cons:**
- OS jobs < 1% of jobs out there

**Pros:**
- Course concepts apply to building correct, high-performance applications (clients of OSes), and building reusable platforms
- Course goes far beyond OS design: computer organization, concurrency, synchronization, input/output, filesystems, networking, routing, distributed systems and so forth
- Don’t be a poser

*The ayes have it!*

Fact

- There has never been as exciting a time to work on systems hardware and software as now!!!
- World is increasingly dependent on computer systems
  - Connected, networked, interlinked
- People don’t know how to build robust systems
Therac-25

- A safety-critical system with software interlocks
- Beam controlled entirely through a custom OS
Therac-25

• Old system used a hardware interlock
  ▪ Lever could either be in “zap” or “x-ray” position
• New system was computer controlled
• A synchronization failure was triggered when competent nurses used the back arrow to change the data on the screen “too quickly”

Therac-25 Outcome

• Beam killed one person directly, burned others, and may have given inadequate treatment to cancer patients
• Problem was very difficult to diagnose; initial fix involved removal of the back arrow key from the keyboard
• People died because a programmer could not write correct code for a concurrent system

Denver airport

• The most modern, most expensive (>>$2B) airport in recent history
• Highly automated luggage handling system was supposed to deliver your luggage to you at arrival
• Lack of persistence caused luggage carts to “forget” their contents, sprinkling the luggage on the runway

USS Yorktown

• Modern guided missile cruiser off of Norfolk, VA in 1998
• Crew member entered 0 for a data value
• Cascade of failures led to a failure of the propulsion system
• Yorktown was dead in the water for a couple of hours
(Concurrency) Bugs in Databases

<table>
<thead>
<tr>
<th>Bug category</th>
<th>DB2 2/03-8/06</th>
<th>Oracle 7/06-11/06</th>
<th>MySQL 8/06-11/06</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBMS crash</td>
<td>120</td>
<td>21</td>
<td>60</td>
</tr>
<tr>
<td>Non-crash faults</td>
<td>131</td>
<td>28</td>
<td>64</td>
</tr>
<tr>
<td>Incorrect answers</td>
<td>81</td>
<td>24</td>
<td>63</td>
</tr>
<tr>
<td>DB corruption</td>
<td>40</td>
<td>4</td>
<td>(inc. above)</td>
</tr>
<tr>
<td>Unauth. access</td>
<td>10</td>
<td>unknown</td>
<td>1</td>
</tr>
</tbody>
</table>

Mostly “independent” Vandiver 2007

Failures in the Cloud

- April 21-25, 2011: Amazon’s EBS down due to operator error and various Mandelbugs
- July 2, 2009 (6 hours): Google’s GFS Mandelbug causes outage of App Engine

“Mandelbugs account for between 15 and 80 percent of all software faults detected after release”


Many machines ➔ many faults

Bitcoin Banks

- There is a new crop of systems for holding data known as NoSQL databases
- Folks @ Cornell and in OS community cautioned about the errors people were committing with NoSQL systems
- Simplest attack—simultaneous withdrawals—actually works! Attackers stole millions of dollars from Bitcoin banks
  - One bank lost a million and folded
  - Another lost $500K, took losses

Other Faulty Systems

- FAA air traffic control system
- IRS data management system
- IBM “Microkernel”
- Pentagon data security
- Many others, too numerous to list
  - Google “software bug” on the News tab...
Summary

• We lack the necessary technologies and know-how to build robust computer systems

• There is huge demand for people who deeply understand and can build robust systems

Philosophy

• Not your grandparents’ OS course
• We believe that the following topics are critical for every software engineer
  ▪ Building concurrent systems
  ▪ Architecting networked components
  ▪ Building transactional systems
• Not widely shared, the course is unique

Logistics

• Lectures
  ▪ Tuesday, Thursday 2:55-4:10pm, Ives 305
  ▪ 4411: Friday 3:35-4:25pm (~every 2 wks), Gates G01
• Instructors
  ▪ Prof. Bracy, bracy at cs.cornell.edu
  ▪ Prof. van Renesse, robbert.vanrenesse at cornell.edu
• Office Hours
  ▪ Prof. van Renesse: ???
  ▪ Prof. Bracy: Wed/Fri 9-10am
• TAs – a small army at your disposal

Communication

• Web Page:
  http://www.cs.cornell.edu/Courses/cs4410
  ▪ Office hours, assignments, lectures, and other supplemental materials will be on the web site
• Piazza: http://piazza.com/cornell/fall2015/cs4410
  ▪ 95% of communication
  ▪ Public posts: for everyone
  ▪ Private posts: for instructor/TA eyes only
• CMS
  ▪ Dissemination of code, grades
Administrative

• Course has three components
  ▪ Lectures and Readings
  ▪ Exams
  ▪ MiniProjects (4410) and Projects (4411)

• You are expected to keep up with all three

• Textbook
  ▪ Anderson and Dahlin (2nd or 1st edition)

Grading

• CS4410 Breakdown
  ▪ ~40% MiniProjects
  ▪ ~12% Prelim 1
  ▪ ~18% Prelim 2
  ▪ ~25% Final
  ▪ ~5% Flexgrade (participation, attitude, effort)

• CS4411 Breakdown
  ▪ ~90% Projects
  ▪ ~10% Flexgrade

• Grading will not be done on a curve
  ▪ We want to give everyone an A+
  ▪ Help us achieve this

MiniProjects in 4410

• 4 miniprojects

• To be done individually

• Start early, time management is key

Projects in CS4411

• Projects done in 2-person teams
  ▪ You may indicate a desired partner
  ▪ If they also indicate you, we will pair you up
  ▪ No Partner? we’ll pair you up with someone suitable on the first Friday

• Working in groups
  ▪ Start early, time management is key
  ▪ Manage the team effort
  ▪ Part of what you are supposed to learn is how to manage to get work done in a small team
Academic Integrity and Honor Code

• All submitted work must be your own
  ▪ All homework must be your own independent work
  ▪ OK to study together
  ▪ Cannot share solutions, ever

• Project groups submit joint work
  ▪ All group assignments must represent solely the work of the two people in that group
  ▪ Cannot be in possession of someone else’s solution

• Violations will be prosecuted to the fullest extent

• Closed-book exams, no calculators

Our Expectations

• Code of Silence
  ▪ Absolute quiet during lectures
  ▪ If you have a question, please speak up
    ▪ Chances are 100% that someone else has the same question

• No electronics, Luddite zone
  ▪ Scientific studies show that such classrooms are far more effective
  ▪ You learn more, so do the people around you

Enrollments

• Goal: Everyone who wants to take 4410 should

• Size limit: this room only holds 318 people

• Enrolled? Congrats!
  ▪ If dropping, please do soon so others can step in!

• Not enrolled? Place your name on waitlist
  ▪ The department—not the instructors—manages the waitlist, sends enrollment PINs

• Good News: 4410 offered Spring 2016

Prerequisite

• CS3410 or equivalent required. Or permission of instructor.

• Required means required.

• If you did not take CS3410 or equivalent, you must contact course staff, explain your situation and request permission.
Tentative Syllabus

- Introduction
- Architectural Support for Operating Systems
- Processes and Threads
- Scheduling
- Synchronization, Mutual Exclusion, Spin Locks, Semaphores, Condition Variables
- Deadlocks, Detection and Avoidance
- Memory Management
- Networking, LANs, WANs, Ethernet, ARP, IP, UDP, TCP
- Disks and RAID
- Filesystems, UFS, LFS
- Security
- Virtualization

Tentative Project Plan for CS4411

- Threads and Concurrency
- Scheduling
- Basic Datagram Networking
- Reliable Streaming Protocols
- Routing
- Filesystems

Questions?