Operating Systems

CS 4410 – CS 4411

Lorenzo Alvisi  Anne Bracy

Spring 2017

These slides build upon many rounds of teaching CS 4410 by Professors Sirer, Bracy, van Renesse, George, Agarwal

About Prof. Bracy

Professional Interests

- Teaching: intro to programming, digital design, computer architecture, system software, operating systems
- Research: microarchitecture, instruction fusion

Past:

- Educated @ Stanford & University of Pennsylvania
- Worked @ WashU in St. Louis & Intel Labs

Other pursuits: novice skier, intermediate jazz connoisseur, advanced toddler wrangler

About Prof. Alvisi

Research interests: building scalable distributed systems that can be depended upon

- PC Chair of SOSP ’17

Undergrad in Physics at ; Ph.D. in CS at

Taught at

Other pursuits: motorcycling, classical music, traveling

About You
Meet the OS

- Software that manages a computer’s resources
- Makes it easier to write the applications you want to write
- Makes you want to use the applications you wrote by running them efficiently

Why study Operating Systems?

- To learn how to manage complexity through appropriate abstractions
  - infinite CPU, infinite memory, files, locks, etc.
- To learn about design
  - performance vs. robustness, functionality vs. simplicity, HW vs. SW, etc.
- To learn how computers work
- Because OSs are everywhere!

Where’s the OS?
Las Vegas
Where’s the OS?
New York

What will the course be like?
Cambia, Todo Cambia

<table>
<thead>
<tr>
<th></th>
<th>1981</th>
<th>1996</th>
<th>2011</th>
<th>Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIPS</td>
<td>1</td>
<td>300</td>
<td>10000</td>
<td>10K</td>
</tr>
<tr>
<td>$/MIPS</td>
<td>$100K</td>
<td>$30</td>
<td>$0.50</td>
<td>200K</td>
</tr>
<tr>
<td>DRAM</td>
<td>128KB</td>
<td>128MB</td>
<td>10GB</td>
<td>100K</td>
</tr>
<tr>
<td>Disk</td>
<td>10MB</td>
<td>4GB</td>
<td>1TB</td>
<td>100K</td>
</tr>
<tr>
<td>Home Internet</td>
<td>9.6Kbps</td>
<td>256 Kbps</td>
<td>5Mbps</td>
<td>500</td>
</tr>
<tr>
<td>LAN Network</td>
<td>3Mbps (shared)</td>
<td>10 Mbps</td>
<td>1Gbps</td>
<td>300</td>
</tr>
<tr>
<td># Users</td>
<td>100</td>
<td>100 Mb/s</td>
<td>&lt;&lt;1</td>
<td>100+</td>
</tr>
</tbody>
</table>

Painting*

- Order
- Design
- Tension
- Balance
- Harmony

System building

- Reliability
- Availability
- Portability
- Efficiency
- Security

*Sondheim: Sunday in the Park with George
Logistics

Lectures
- 4410: Tu-Th 1:25-2:40pm, Olin 155
- 4411: F 2:30-3:30pm, Hollister B14 (~ every 2 weeks)

Instructors:
- Professor Alvisi: M: 6:00-8:00pm
- Professor Bracy: M: 11:00-12:00pm · Tu: 3:00-4:00pm
- TAs — a small army at your disposal!

Office Hours

Our Expectations

Code of Silence
- Absolute quiet during lectures
- Except (duh!) for questions! Please ask!

Luddite Zone
- Numerous studies show that such classrooms are far more effective (pioneered by Cornell: “The Laptop and the Lecture”, 2003)
- You learn more, so do the people around you!

Communicating

Web Page: http://www.cs.cornell.edu/Courses/cs4410
- Office hours, assignments, lectures, and other supplemental materials will be on the web site
- Public posts: for everyone
  - Don’t post code
  - Use posts, not email
- Private posts: for instructor/TA eyes only
- Personal emergencies: email cs4410-prof@cornell.edu (goes to us both)

Assignments/Projects

Code distributed through github
- http://github.coecis.cornell.edu
- we’ll need your ids — more on this later

Submissions through CMS

Administrative

- You are expected to keep up with
  - Lectures and Readings
  - Exams
  - Assignments (4410) and Projects (4411)
- Textbook
  - Anderson and Dahlin (1st or 2nd edition)
  - Subset of Kurose and Ross “Computer Networking: A Top-Down Approach”.
Grading

CS4410
- 40% Programming Assignments
- 12% Prelim 1
- 12% Prelim 2
- 25% Final
- 5% Flexgrade

CS4411
- 90% Projects
- 10% Flexgrade

Grading will not be on a curve
- We want to give everyone an A+
- It's a time trial: you are not running against your peers

Programming Assignments (CS4410)

- 4 assignments
  - Shell
  - Concurrent programming
  - Networking
  - File systems

- To be done individually
- 4 slip days — at most 2 per assignment


tart early! Time management is key

Projects (CS4411)

- 6 project, to be done in teams of 2
  - Threads and Concurrency
  - Basic Datagram Networking
  - Routing
  - Scheduling
  - Reliable Streaming Protocol
  - File Systems

- Google form to indicate team's composition
  - No partner? We've got a Google form for that too! Or search on Piazza

- Working in pairs
  - Start early; time management is key; Manage the team effort
  - Don't let your team member down

- 4 slip days — at most 1 per assignment

Academic Integrity and Honor Code

All submitted work must be your own (CS4410) or your group’s (CS4411)

- Project groups submit joint work
  - All programming assignments must be your own independent work
  - Group projects must represent solely the work of the two members of the group
  - OK to study together (with the Game of Thrones rule) but never look at someone else's code (fellow student, or online, or...)

- Violations are easy to detect & will be prosecuted

- Closed book exams, no calculators
Prerequisites

- We are checking your prerequisites
  - informally CS3410; or cs3420; or equivalent course on “Architecture & Systems Software”
- If you don’t meet them, we’ll contact you

Questions?

Running a Web Server

- How does the OS
  - allow multiple applications to communicate with each other?
  - handle multiple concurrent requests?
  - support access to shared data (such as the cache)?
  - protect against malicious scripts?
  - enable different apps to share the data they have produced?
  - support consistent changes to complex data structures?
  - handle clients and servers of different speed?
  - transparently move to more powerful hardware?

Course objective
Leg 1

1. Learn how to approach complex problems
   - Fundamental issues
   - coordination, abstraction
   - Explore design space
   - Examine case studies
   - Goal: Forever mutate your brain (Mwahahahaahhaha!)
   - Timescale: Big, long-term payoff

Leg 2

2. Learn how to apply specific techniques
   - Debug complex systems
   - Time-tested solutions to hard problems
   - Hacking will not succeed
     - concurrent programming, transactions, etc
   - Goal: Be a good engineer
   - Timescale: Now — and in 20 years

Leg 3

3. Learn how, in detail, current OSs work
   - FS, network stack, internal data structures, VM... of
     - MacOS, Linux, iOS, Windows
   - Goal: Well... know in detail how current OSs work!
   - Timescale: Better be now, because all will change tomorrow

What is an OS?

- An Operating System implements a virtual machine whose interface is more convenient* that the raw hardware interface

* easier to use, simpler to code, more reliable, more secure...

“All the code you did not write”
More than one hat

- Referee
  - Manages shared resources such as CPU, memory, disks, networks, displays, cameras, etc.

- Illusionist
  - Look! Infinite memory! Your own private processor!

- Glue

More than one hat

- Referee

- Illusionist

- Glue

Offers a set of common services (e.g. U.I. routines)
Separates apps from I/O devices
**OS as a referee**

- Resource allocation
  - When multiple concurrent tasks, how does OS decide who gets how much?

- Isolation
  - A faulty app should not disrupt other apps or OS
    - OS must export less than full power of underlying hardware

- Communication/Coordination
  - Apps need to coordinate and share state
    - Web site: select ads, cache recent data, fetch/merge data from disk, etc

**OS as an illusionist**

- Illusion of resources that are not physically present
  - Virtualization
    - Processor, memory, screen space, disk, network

- Atomic operations
  - Hardware guarantees atomicity at the word level
  - What happens during concurrent updates to complex data structures?
  - What if computer crashes during a block write?
  - At the hardware level, packets are lost...

- Reliable communication channels
OS as a glue

- Offers standard services to simplify app design and facilitate sharing
  - send/receive of byte streams
  - read/write files
  - pass messages
  - share memory
  - UI
- Decouples hardware and app development
  - ...but database may need to be aware of specific disk drive

What makes a good OS?

The right set of abstractions

A good abstraction:

- is portable and hides implementation details
- has an intuitive and easy-to-use interface
- can be installed many times
- is efficient and reasonably easy to implement

OS: a collection of abstractions

- Processes (abstract CPU and RAM)
- Files (abstract disks)
- Network endpoints (abstract NIC)
- Windows (abstract screens)
- ...

Think of them as objects with state and methods

Issues in OS Design

- Structure: how is the 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 it fast?
More issues in OS Design

- **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 bill and how do we control resource usage?

A Short History of Operating Systems

History of Operating Systems

- **Phase 1:** Hardware is expensive, humans are cheap
  - User at console: single-user systems
  - Batching systems
  - Multi-programming systems

Hand Programmed machines (1945-1955)

- Single user systems
- OS = loader + libraries of common subroutines
- Problem: low utilization of expensive components

\[
\text{time device busy} \times \frac{\text{observation interval}}{\text{time interval}} = \% \text{ utilization}
\]
**Batch Processing (1955-1965)**

Operating system = loader + sequencer + output processor

- **Input**
  - Card Reader
  - Tape

- **Compute**
  - User Program
  - User Data
  - “System Software”
  - Operating System

- **Output**
  - Printer
  - Tape

**Multiprogramming (1965-1980)**

Keep several jobs in memory and multiplex CPU between jobs

- **Program 1**
  - User Program 1
  - User Program 2
  - “System Software”
  - Operating System
  - I/O Device

- **Program 2**
  - User Program 1
  - User Program 2
  - “System Software”
  - Operating System
  - I/O Device

**Program 1**

```
main()
k: read()
    startIO()
    waitIO()
endio()
k+1:
```

**Program 2**

```
main()
    read()
    startIO()
    schedule()
endio()
```

**OS call Read()**

```
program P
begin
  Read(var)
end P
```

```
program P
begin
  Read(var)
end P
```

```
main{
    schedule()
}
```

```
main{
    schedule()
}
```

```
interrupt
```

```
interrupt
```
History of Operating Systems

- **Phase 1**: Hardware is expensive, humans are cheap
  - User at console: single-user systems
  - Batching systems
  - Multi-programming systems
- **Phase 2**: Hardware is cheap, humans are expensive
  - Time sharing: Users use cheap terminals and share servers
- **Phase 3**: Hardware is very cheap, humans are very expensive
  - Personal computing: One system per user
  - Distributed computing: many systems per user
  - Ubiquitous computing: LOTS of systems per users

History of Operating Systems

- **Phase 1**: Hardware is expensive, humans are cheap
  - User at console: single-user systems
  - Batching systems
  - Multi-programming systems
- **Phase 2**: Hardware is cheap, humans are expensive
  - Time sharing: Users use cheap terminals and share servers
- **Phase 3**: Hardware is very cheap, humans are very expensive
  - Personal computing systems
  - Utilization no longer a concern
  - Emphasis on user interface and API
  - Many services & features not present

Evolution

- Initially: OS as a simple service provider (simple libraries)
- Now: Multi-application systems with support for coordination

Timesharing (1970-)

A timer interrupt is used to multiplex CPU between jobs

Operating Systems for PCs

- Personal computing systems
  - Single user
  - Utilization no longer a concern
  - Emphasis on user interface and API
  - Many services & features not present

Evolution

- Initially: OS as a simple service provider (simple libraries)
- Now: Multi-application systems with support for coordination
Distributed Operating Systems

- Abstraction: a multi-processor system as a single processor one.
- New challenges in consistency, reliability, resource management, performance, etc.
- Examples: SANs, Oracle Parallel Server

User Program

- OS
  - process management
  - memory management

CPU

Network

User Program

- OS
  - process management
  - name services
  - mail services

OS

CPU

Ubiquitous Computing

- Challenges
  - Small memory size
  - Slow processor
  - Battery concerns
  - Scale
  - Security
  - Naming