COS 318: Operating Systems

Introduction

Jaswinder Pal Singh
Computer Science Department
Princeton University

http://www.cs.princeton.edu/courses/archive/fall16/cos318/
Today

- Course information and logistics
- What is an operating system?
- Evolution of operating systems
- Why study operating systems?
Information and Staff

◆ Website

◆ Textbooks

◆ Instructors
  ● Jaswinder Pal Singh, Office: 423 CS, Hours: Mon 1:30 – 3 pm

◆ Teaching assistants (offices and hours to be posted on web site)
  ● Qizhe Cai,
  ● Ghassan Jerfel
  ● Pranjit Kalita
  ● Huilian (Sophie) Qiu
Grading

- Projects 60%
- Final project 20%
- Exam 20%

- No final exam after break
Projects

Projects
- Bootloader (150-300 lines)
- Non-preemptive kernel (200-250 lines)
- Preemptive kernel (100-150 lines)
- Inter-process communication and device driver (300-350 lines)
- Virtual memory (300-450 lines)
- File system (500+ lines)

How
- Pair with a partner for project 1, 2 and 3
- Pair with a different partner for project 4 and 5
- Do the final project yourself (no partners)
- Design review at the end of week one
- All projects due Sundays at 11:55pm

Where to do the projects
- Develop on courselab machines, via remote login from your computer
- Create bootable image on USB drive
- Test using bochs, final test on lab machines in Friend 010
Project Grading

◆ Design Review
  ● Requirements will be specified for each project
  ● Sign up online for making appointments
  ● 10 minutes with the TA in charge
  ● 0-5 points for each design review
  ● 10% deduction for missing an appointment

◆ Project completion
  ● Assigned project points plus possible extra points

◆ Late policy for grading projects
  ● 1 hour: 98.6%, 6 hours: 92%, 1 day: 71.7%
  ● 3 days: 36.8%, 7 days: 9.7%
Logistics

- Precepts
  - Time: Mon 7:30pm – 8:20pm in CS 105
  - No second session

- For project 1
  - A tutorial on assembly programming and kernel debugging
    - Mon 9/19: 7:30-8:20pm in CS 105
  - Precept
    - 9/26: 7:30-8:20pm in CS 105
  - Design review
    - 9/26 (Monday) 1:30pm – evening (Friend 010)
    - Sign up online (1 slot per team)
  - Due: 10/2 (Sunday) 11:55pm
Piazza for Discussions

- Piazza is convenient
  - Most of you love it (?)

- Search, ask and answer questions
  - Students are encouraged to answer questions
  - Staff will try to answer in a timely manner

- Only use email if your question is personal/private
  - Project grading questions: send email to the TA in charge
Ethics and Other Issues

- Follow Honor System
  - Ask teaching staff if you are not sure
  - Asking each other questions is okay
  - **Work must be your own (or your team’s)**
- If you discover any solutions online
  - Tell teaching staff
- Do not put your code or design on the web, in social media, or anywhere public or available to others ...
COS318 in Systems Course Sequence

- **Prerequisites**
  - COS 217: Introduction to Programming Systems
  - COS 226: Algorithms and Data Structures
- **300-400 courses in systems**
  - COS318: Operating Systems
  - COS320: Compiler Techniques
  - COS333: Advanced Programming Techniques
  - COS432: Information Security
  - COS475: Computer Architecture
- **Courses needing COS318**
  - COS 461: Computer Networks
  - COS 518: Advanced Operating Systems
  - COS 561: Advanced Computer Networks
What Is Operating System?

- Software between applications and hardware
- Provide abstractions to layers above
- Implement abstractions for and manage resources below
- What about the UI?
Consider reading from disks

- Different types of disks, with very different structures
  - Floppy, various kinds of hard drives, Flash, IDE, ...
- Different hardware mechanisms to read, different layouts of data on disk, different mechanics

- Floppy disk has ~20 commands to interact with it
- Read/write have 13 parameters; controller returns 23 codes
- Motor may be on or off, don’t read when motor off, etc.
- And this is only one disk type

- Rather, a simple abstraction: data are in files, you read from and write to files using simple interfaces
- OS manages all the rest
What Do Operating Systems Do?

- Provides abstractions to user-level software above
  - User programs can deal with simpler, high-level concepts
  - Hide complex and unreliable hardware, and the variety of hardware
  - Provide illusions like “sole application running” or “infinite memory”

- Implement the abstractions: manage resources
  - Manage application interaction with hardware resources
  - Allow multiple applications and multiple users to share resources effectively without hurting one another
  - Protect application software from crashing a system
Some Examples

- System example
  - What if a user tries to access disk blocks directly?

- Protection example
  - What if a user program can access all RAM memory?
  - What if a user runs the following code:
    ```c
    int main() {
        while(1)
            fork();
    }
    ```

- Resource management example
  - What if many programs are running infinite loops?
    ```c
    while (1);
    ```
## A Typical Academic Computer (1981 vs. 2011)

<table>
<thead>
<tr>
<th>Feature</th>
<th>1981</th>
<th>2011</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel CPU transistors</td>
<td>0.1M</td>
<td>1.9B</td>
<td>~20000x</td>
</tr>
<tr>
<td>Intel CPU core x clock</td>
<td>10Mhz</td>
<td>10×2.4Ghz</td>
<td>~2,400x</td>
</tr>
<tr>
<td>DRAM</td>
<td>1MB</td>
<td>64GB</td>
<td>64,000x</td>
</tr>
<tr>
<td>Disk</td>
<td>5MB</td>
<td>1TB</td>
<td>200,000x</td>
</tr>
<tr>
<td>Network BW</td>
<td>10Mbits/sec</td>
<td>10GBits/sec</td>
<td>1000x</td>
</tr>
<tr>
<td>Address bits</td>
<td>32</td>
<td>64</td>
<td>2x</td>
</tr>
<tr>
<td>Users/machine</td>
<td>10s</td>
<td>&lt;1</td>
<td>&gt;10x</td>
</tr>
<tr>
<td>$/machine</td>
<td>$30K</td>
<td>$1.5K</td>
<td>1/20x</td>
</tr>
<tr>
<td>$/Mhz</td>
<td>$30,000</td>
<td>$1,500/24,000</td>
<td>1/4,800x</td>
</tr>
</tbody>
</table>
Exponential Growth in Computing and Communications (Courtesy Jim Gray)

- #transistors on chip doubles every 18 months
- 100x per decade
- Progress in next 18 months = ALL previous progress
  - New storage = sum of all past storage (ever)
  - New processing = sum of all past processing power
  - Bandwidth at even faster pace

15 years ago
Phase 1: Hardware Expensive, Human Cheap

- User at console, OS as subroutine library
- Batch monitor (no protection): load, run, print
- A lot of the (expensive) hardware sits idle a lot. Developments:
  - Interrupts; overlap I/O and CPU
  - Direct Memory Access (DMA)
  - Memory protection: keep bugs to individual programs
  - Multics: designed in 1963 and run in 1969; multiprogramming
- Assumption: No bad people. No bad programs. Minimum interactions
Phase 2: Hardware Cheap, Human Expensive

- Use cheap terminals to share a computer
- Time-sharing OS
- Unix enters the mainstream as hardware got cheaper
- Problems: thrashing as the number of users increases
Phase 3: HW Cheaper, Human More Expensive

- **Personal computer**
  - Altos OS, Ethernet, Bitmap display, laser printer (79)
  - Pop-menu window interface, email, publishing SW, spreadsheet, FTP, Telnet
  - Became >200M units per year

- **PC operating system**
  - Memory protection
  - Multiprogramming
  - Networking

*First PC at Xerox PARC*
Now: > 1 Machines per User

- Pervasive computers
  - Wearable computers
  - Communication devices
  - Entertainment equipment
  - Computerized vehicle
  - Phones ~2B units /year

- OS are specialized
  - Embedded OS
  - Specially general-purpose OS (e.g. iOS, Android)
Now: Multiple Processors per “Machine”

- **Multiprocessors**
  - SMP: Symmetric MultiProcessor
  - ccNUMA: Cache-Coherent Non-Uniform Memory Access
  - General-purpose, single-image OS with multiprocessor support

- **Multicomputers**
  - Supercomputer with many CPUs and high-speed communication
  - Specialized OS with special message-passing support

- **Clusters**
  - A network of PCs
  - Server OS w/ cluster abstraction (e.g. MapReduce)
Now: Multiple “Cores” per Processor

- Multicore or Manycore transition
  - Intel Xeon processor has 10 cores / 20 threads
  - New Intel Xeon Phi has 60 cores
  - nVidia GPUs has 3000 FPUs

- Accelerated need for software support
  - OS support for manycores
  - Parallel programming of applications
Now: Datacenter as A Computer

- **Cloud computing**
  - Hosting data in the cloud
  - Software as services
  - Examples:
    - Google, Microsoft, Salesforce, Yahoo, …

- **Utility computing**
  - Pay as you go for computing resources
  - Outsourced warehouse-scale hardware and software
  - Examples:
    - Amazon, Google, Micros
Why Study OS?

- OS is a key part of a computer system
  - It makes our life better (or worse)
  - It is “magic” to realize what we want
  - It gives us “power” (reduce fear factor)
- Learn how computer systems really work, who does what, how
- Learn key CS concepts: abstraction, layering, virtualization, indirection
- Learn about concurrency
  - Parallel programs run on OS
  - OS runs on parallel hardware
  - Best way to learn concurrent programming
- Understand how a system works
  - How many procedures does a key stroke invoke?
  - What happens when your application references 0 as a pointer?
Why Study OS?

- Basic knowledge for many areas
  - Networking, distributed systems, security, …
- Build an OS
  - Real OS is huge, but building a small OS will go a long way
- More employable
  - Become someone who understand “systems”
  - Become the top group of “athletes”
  - Ability to build things from ground up

Question:
- Why shouldn’t you study OS?
Does COS318 Require A Lot of Time?

- Yes
  - But less than a couple of years ago, and we’re trying to make it even less
  - Part of that is measuring where time goes (see later)
- To become a top athlete, you want to know the entire HW/SW stack, and spend 10,000 hours programming
  - “Practice isn't the thing you do once you're good. It's the thing you do that makes you good.”
  - “In fact, researchers have settled on what they believe is the magic number for true expertise: ten thousand hours.”
    — Malcolm Gladwell, *Outliers: The Story of Success*
Things to Do

- Today’s material
  - Read MOS 1.1-1.3
  - Lecture available online

- Next lecture
  - Read MOS 1.4-1.5

- Make “tent” with your name
  - Use next time

- Use piazza to find a partner
  - Find a partner before the end of next lecture for projects 1, 2 and 3