CS162
Operating Systems and
Systems Programming
Lecture 1

What is an Operating System?

January 16th, 2024
Prof. John Kubiatowicz
http://cs162.eecs.Berkeley.edu

Goals for Today

- What is an Operating System?
  - And what is it not?
- What makes Operating Systems so exciting?
- Oh, and “How does this class operate?”

Interactive is important!
Ask Questions!

Greatest Artifact of Human Civilization…

The Internet!
Running Systems at Internet Scale

And Range of Timescales

Jeff Dean:
"Numbers Everyone Should Know"

Operating Systems are at the Heart of it All!

- Make the incredible advance in the underlying technology available to a rapidly evolving body of applications
  - Provide **consistent abstractions** to applications, even on different hardware
  - Manage **sharing of resources** among multiple applications

- The key building blocks:
  - Processes
  - Threads, Concurrency, Scheduling, Coordination
  - Address Spaces
  - Protection, Isolation, Sharing, Security
  - Communication, Protocols
  - Persistent storage, transactions, consistency, resilience
  - Interfaces to all devices
Example: What’s in a Search Query?

- Complex interaction of multiple components in multiple administrative domains
  - Systems, services, protocols, ...

But: What is an operating system?

What does an Operating System do?

- Most Likely:
  - Memory Management
  - I/O Management
  - CPU Scheduling
  - Communications? (Does Email belong in OS?)
  - Multitasking/multiprogramming?
- What about?
  - File System?
  - Multimedia Support?
  - User Interface?
  - Internet Browser?
- Is this only interesting to Academics??

Definition of an Operating System

- No universally accepted definition
- “Everything a vendor ships when you order an operating system” is good approximation
  - But varies wildly
- “The one program running at all times on the computer” is the **kernel**
  - Everything else is either a system program (ships with the operating system) or an application program
One Definition of an Operating System

- Special layer of software that provides application software access to hardware resources
  - Convenient abstraction of complex hardware devices
  - Protected access to shared resources
  - Security and authentication
  - Communication

Operating System

What makes something a system?

- Multiple interrelated parts
  - Each potentially interacts with the others
- Robustness requires an engineering mindset
  - Meticulous error handling, defending against malicious careless users
  - Treating the computer as a concrete machine, with all of its limitations and possible failure cases

System programming is an important part of this class!
What is an Operating System?

- **Illusionist**
  - Provide clean, easy-to-use abstractions of physical resources
    - Infinite memory, dedicated machine
    - Higher level objects: files, users, messages
    - Masking limitations, virtualization

OS Basics: Virtualizing the Machine

Compiled Program’s View of the World

System Programmer’s View of the World

- Application’s “machine” is the process abstraction provided by the OS
- Each running program runs in its own process
- Processes provide nicer interfaces than raw hardware
What's in a Process?

A process consists of:
- Address Space
- One or more threads of control executing in that address space
- Additional system state associated with it
  - Open files
  - Open sockets (network connections)
  - ...

For Example...

Operating System’s View of the World

- OS translates from hardware interface to application interface
- OS provides each running program with its own process
What is an Operating System?

- **Referee**
  - Manage protection, isolation, and sharing of resources
    - Resource allocation and communication
- **Illusionist**
  - Provide clean, easy-to-use abstractions of physical resources
    - Infinite memory, dedicated machine
    - Higher level objects: files, users, messages
    - Masking limitations, virtualization

OS Basics: Running a Process

OS Basics: Switching Processes
OS Basics: Switching Processes

Process 1
- Compiled Program 1
- System Libs
- Process
- Threads
- Address Spaces
- Files
- Sockets

Process 2
- Compiled Program 2
- System Libs
- Process
- Threads
- Address Spaces
- Files
- Sockets

Operating System
- Processor
- Memory
- Storage
- Networks
- Hardware

ISA

OS Basics: Protection

Process 1
- Compiled Program 1
- System Libs
- Process
- Threads
- Address Spaces
- Files
- Sockets

Process 2
- Compiled Program 2
- System Libs
- Process
- Threads
- Address Spaces
- Files
- Sockets

Operating System
- Processor
- Memory
- Storage
- Networks
- Hardware

ISA

Segmentation fault (core dumped)
OS Basics: Protection

- OS isolates processes from each other
- OS isolates itself from other processes
- ... even though they are actually running on the same hardware!

OS Basics: I/O

- OS provides common services in the form of I/O

OS Basics: Look and Feel

What is an Operating System?

- Referee
  - Manage protection, isolation, and sharing of resources
    » Resource allocation and communication
- Illusionist
  - Provide clean, easy-to-use abstractions of physical resources
    » Infinite memory, dedicated machine
    » Higher level objects: files, users, messages
    » Masking limitations, virtualization
- Glue
  - Common services
    » Storage, Window system, Networking
    » Sharing, Authorization
    » Look and feel
OS Basics: Background Management

What is an Operating System?

• Referee
  – Manage protection, isolation, and sharing of resources
    » Resource allocation and communication

• Illusionist
  – Provide clean, easy-to-use abstractions of physical resources
    » Infinite memory, dedicated machine
    » Higher level objects: files, users, messages
    » Masking limitations, virtualization

• Glue
  – Common services
    » Storage, Window system, Networking
    » Sharing, Authorization
    » Look and feel

Why take CS162?

• Some of you will actually design and build operating systems or components of them.
  – Perhaps more now than ever

• Many of you will create systems that utilize the core concepts in operating systems.
  – Whether you build software or hardware
  – The concepts and design patterns appear at many levels

• All of you will build applications, etc. that utilize operating systems
  – The better you understand their design and implementation, the better use you'll make of them.

Who am I? John Kubiatowicz (Prof “Kubi”)

• Background in Hardware Design
  – Alewife project at MIT
  – Designed CMMU, Modified SPAR C processor
  – Helped to write operating system

• Background in Operating Systems
  – Worked for Project Athena (MIT)
  – OS Developer (device drivers, network file systems)
  – Worked on Clustered High-Availability systems.

• Peer-to-Peer
  – OceanStore project – Store your data for 1000 years
  – Tapestry and Bamboo – Find you data around globe
  – One of the first cloud storage projects! (Before the cloud!)

• Quantum Computing
  – Exploring architectures for quantum computers
  – CAD tool set yields some interesting results

• SwarmLAB/Berkeley Lab for Intelligent Edge
  – Global Data Plane (GDP)/DataCapsules
  – Fog Robotics
  – Hardened Data Containers
CS162 TAs: Sections TBA

Enrollment

• This term class size is limited for funding reasons
  – We expanded the class last week to 13 sections and 404 students
    » No more expansion
    » Replacements will come off the waitlist
  – Please do not email us asking for special reordering of the waitlist!
    » Ordering is dictated by department policy!
• This is an Early Drop Deadline course (January 26th)
  – If you are not serious about taking this class, please drop quickly
    » Department will continue to admit students as other students drop
  – Really hard to drop afterwards!
    » Don’t forget to keep up with work if you are still on the waitlist!
• As long as you are on the waitlist or applying for concurrent enrollment, you must do the work!
  – If you are no longer interested in the course, please remove yourself from waitlist

Infrastructure, Textbook & Readings

• Infrastructure
  – Website: https://cs162.org
  – Lecture Recordings: Tentatively as links off main class page (within one week?)
  – Suggested readings posted along with lectures
  – Try to keep up with material in book as well as lectures
• Supplementary Material
  – Operating Systems: Three Easy Pieces, by Remzi and Andrea Arpaci-Dusseau, available for free online
  – Linux Kernel Development, 3rd edition, by Robert Love
• Online supplements
  – See course website
  – Includes Appendices, sample problems, etc.
  – Networking, Databases, Software Eng, Security
  – Some Research Papers!

Syllabus

• OS Concepts: How to Navigate as a Systems Programmer!
  – Process, I/O, Networks and Virtual Machines
• Concurrency
  – Threads, scheduling, locks, deadlock, scalability, fairness
• Address Space
  – Virtual memory, address translation, protection, sharing
• File Systems
  – I/O devices, file objects, storage, naming, caching, performance, paging, transactions, databases
• Distributed Systems
  – Protocols, N-Tiers, RPC, NFS, DHTs, Consistency, Scalability, multicast
• Reliability & Security
  – Fault tolerance, protection, security
• Cloud Infrastructure
Learning by Doing

- Individual Homeworks (2-3 weeks) - preliminary
  - 0. Tools & Environment, Autograding, recall C, executable
  - 1. Lists in C
  - 2. BYOS – build your own shell
  - 3. Sockets & Threads in HTTP server
  - 4. Memory mapping and management
  - 5. Map Reduce
- Three (and ½) Group Projects
  - 0. Getting Started (Individual, before you have a group)
  - 1. User-programs (exec & syscall)
  - 2. Threads & Scheduling
  - 3. File Systems

Group Projects

- Project teams have 4 members!
  - never 5, 3 requires serious justification
- Must work in groups in “the real world”
  - Same section (at least same TA)
- Communication and cooperation will be essential
  - Regular in-person meetings very important!
  - Joint work on Design Documents
  - Slack/Messenger/whatever doesn’t replace face-to-face!
- Everyone should do work and have clear responsibilities
  - You will evaluate your teammates at the end of each project
  - Dividing up by Task is the worst approach. Work as a team.
- Communicate with supervisor (TAs)
  - What is the team’s plan?
  - What is each member’s responsibility?
  - Short progress reports are required
  - Design Documents: High-level description for a manager!

Getting started

- EVERYONE (even if you are on the waitlist!):
  Start homework 0 right away (hopefully Today!), project 0 next week
  - Github account
  - VM environment for the course
    » Consistent, managed environment on your machine
  - Get familiar with all the cs162 tools
  - Submit to autograder via git
- First two weeks, attend any section you want
  - We’ll assign permanent sections after forming project groups
  - Section attendance will be mandatory after we form groups
  - These section times will be adjusted after we have a better idea where people are

Preparing Yourself for this Class

- The projects will require you to be very comfortable with programming and debugging C
  - Pointers (including function pointers, void*)
  - Memory Management (malloc, free, stack vs heap)
  - Debugging with GDB
- You will be working on a larger, more sophisticated code base than anything you’ve likely seen in 61C!
- Review Session on the C language
  - Time and logistics TBA, but soon!
- "Resources” page on course website
  - Ebooks on “git” and “C"
- C programming reference (still in beta):
  - https://cs162.org/ladder/
- First two sections are also dedicated to programming and debugging review:
  - Attend ANY sections in first two weeks
Grading (Tentative breakdown)

- 36% three midterms (12% each)
  - Thursday, 2/15 (8-10pm), No class on day of Midterm (extra OH)
  - Thursday, 3/14 (8-10pm), No class on day of Midterm (extra OH)
  - Thursday, 4/25 (8-10pm), No class on day of Midterm (extra OH)
  - These will be IN-PERSON!
- 36% projects
- 18% homework
- 10% participation (Sections, Lecture, …)
- No final exam
- Projects
  - Initial design document, Design review, Code, Final design
  - Submission via git push triggers autograder

Personal Integrity

- UCB Academic Honor Code: "As a member of the UC Berkeley community, I act with honesty, integrity, and respect for others."

CS 162 Collaboration Policy

- Explaining a concept to someone in another group
- Discussing algorithms/testing strategies with other groups
- Discussing debugging approaches with other groups
- Searching online for generic algorithms (e.g., hash table)
- Sharing code or test cases with another group
- Copying OR reading another group’s code or test cases
- Copying OR reading online code or test cases from prior years
- Helping someone in another group to debug their code

- We compare all project submissions against prior year submissions and online solutions and will take actions (described on the course overview page) against offenders
- Don’t put a friend in a bad position by asking for help that they shouldn’t give!

Typical Lecture Format

1-Minute Review

20-Minute Lecture

5-Minute Administrative Matters

25-Minute Lecture

5-Minute Break (water, stretch)

25-Minute Lecture

"In Conclusion, …"
Lecture Goal

Interactive!!!
Ask Questions in Chat

What makes Operating Systems Exciting and Challenging?

Societal Scale Information Systems
(Or the “Internet of Things”?)

- The world is a large distributed system
- Microprocessors in everything
- Vast infrastructure behind them

Internet Connectivity
MEMS for Sensor Nets

Scalable, Reliable, Secure Services
Databases
Information Collection
Remote Storage
Online Games
Commerce
...

Technology Trends: Moore's Law

Gordon Moore (co-founder of Intel) predicted in 1965 that the transistor density of semiconductor chips would double roughly every 18 months

Microprocessors have become smaller, denser, and more powerful
Big Challenge: Slowdown in Joy's law of Performance


Performance (vs. VAX-11/780)

- VAX: 25%/year 1978 to 1986
- RISC + x86: 52%/year 1986 to 2002
- RISC + x86: ??%/year 2002 to present

Another Challenge: Power Density

- Moore’s law extrapolation
  – Potential power density reaching amazing levels!
- Flip side: battery life very important
  – Moore’s law yielded more functionality at equivalent (or less) total energy consumption

Sea change in chip design: multiple cores or processors per chip

Sea change in chip design: multiple cores or processors per chip

- Moore’s Law has (officially) ended -- Feb 2016
  – No longer getting 2 x transistors/chip every 18 months…
  – or even every 24 months
- May have only 2-3 smallest geometry fabrication plants left:
  – Intel and Samsung and/or TSMC
- Vendors moving to 3D stacked chips
  – More layers in old geometries

ManyCore Chips: The future arrived in 2007

- Intel 80-core multicore chip (Feb 2007)
  – 80 simple cores
  – Two FP-engines / core
  – Mesh-like network
  – 100 million transistors
  – 65nm feature size
  – 24 “tiles” with two cores/tile
  – 24-router mesh network
  – 4 DDR3 memory controllers
  – Hardware support for message-passing

How to program these?
- Use 2 CPUs for video/audio
- Use 1 for word processor, 1 for browser
- 76 for virus checking???

Parallelism must be exploited at all levels

Amazon X1 instances (2016)
- 128 virtual cores, 2 TB RAM

But then Moore’s Law Ended…
Storage Capacity is Still Growing!

Society is Increasingly Connected…

Network Capacity Still Increasing

Not Only PCs connected to the Internet

- In 2011, smartphone shipments exceeded PC shipments!
- 2011 shipments:
  - 487M smartphones
  - 414M PC clients
    - 210M notebooks
    - 112M desktops
    - 63M tablets
  - 25M smart TVs
- 4 billion phones in the world → smartphones over next few years
- Then…

People-to-Computer Ratio Over Time

- Number crunching, Data Storage, Massive Inet Services, ML, ...
- Productivity, Interactive
- Streaming from/to the physical world

Bell's Law: new computer class per 10 years

The Internet of Things!

What is an Operating System Again?

- Referee
  - Manage sharing of resources, Protection, Isolation
    » Resource allocation, isolation, communication
- Illusionist
  - Provide clean, easy to use abstractions of physical resources
    » Infinite memory, dedicated machine
    » Higher level objects: files, users, messages
    » Masking limitations, virtualization
- Glue
  - Common services
    » Storage, Window system, Networking
    » Sharing, Authorization
    » Look and feel

Challenge: Complexity

- Applications consisting of...
  - ... a variety of software modules that ...
  - ... run on a variety of devices (machines) that
    » ... implement different hardware architectures
    » ... run competing applications
    » ... fail in unexpected ways
    » ... can be under a variety of attacks
- Not feasible to test software for all possible environments and combinations of components and devices
  - The question is not whether there are bugs but how serious are the bugs!

The World Is Parallel: e.g. Intel Saphire Rapids (2023)

- Up to 60 cores, 120 threads/package (socket)
  - Up to 4 “chiplets” bonded together
- Network:
  - On-chip Mesh Interconnect
  - Fast off-chip network (UPI): directly connects 8-chips
  - 480 cores/shared memory domain!
- Each Core Has:
  - 80 KB L1 Cache
  - 2 MB L2 Cache
  - Fraction of up to 112.5 MB L3 Cache
- DRAM/chips
  - Up to 4 TiB of DDR5 memory
- Many Accellerators of different types
  - Graphics, Encryption, AI, Security
HW Functionality comes with great complexity!

Increasing Software Complexity

Example: Some Mars Rover (“Pathfinder”) Requirements

- Pathfinder hardware limitations/complexity:
  - 20Mhz processor, 128MB of DRAM, VxWorks OS
  - cameras, scientific instruments, batteries, solar panels, and locomotion equipment
  - Many independent processes work together
- Can’t hit reset button very easily!
  - Must reboot itself if necessary
  - Must always be able to receive commands from Earth
- Individual Programs must not interfere
  - Suppose the MUT (Martian Universal Translator Module) buggy
  - Better not crash antenna positioning software!
- Further, all software may crash occasionally
  - Automatic restart with diagnostics sent to Earth
  - Periodic checkpoint of results saved?
- Certain functions time critical:
  - Need to stop before hitting something
  - Must track orbit of Earth for communication
- A lot of similarity with the Internet of Things?
  - Complexity, QoS, Inaccessibility, Power limitations … ?

Questions

- Does the programmer need to write a single program that performs many independent activities?
- Does every program have to be altered for every piece of hardware?
- Does a faulty program crash everything?
- Does every program have access to all hardware?

Hopefully, no!

Operating Systems help the programmer write robust programs!
OS Abstracts the Underlying Hardware

- Processor → Thread
- Memory → Address Space
- Disks, SSDs, ... → Files
- Networks → Sockets
- Machines → Processes

OS as an Illusionist:
- Remove software/hardware quirks (fight complexity)
- Optimize for convenience, utilization, reliability, ... (help the programmer)
- For any OS area (e.g. file systems, virtual memory, networking, scheduling):
  - What hardware interface to handle? (physical reality)
  - What's software interface to provide? (nicer abstraction)

Basic Tool: Dual-Mode Operation

- Hardware provides at least two modes:
  1. Kernel Mode (or “supervisor” mode)
  2. User Mode
- Certain operations are prohibited when running in user mode
  - Changing the page table pointer, disabling interrupts, interacting directly with hardware, writing to kernel memory
- Carefully controlled transitions between user mode and kernel mode
  - System calls, interrupts, exceptions

OS Protects Processes and the Kernel

- Run multiple applications and:
  - Keep them from interfering with or crashing the operating system
  - Keep them from interfering with or crashing each other

UNIX System Structure

User Mode
- Applications (the users)
- Standard Libs (shells and commands, compilers and interpreters, system libraries)
- system-call interface to the kernel

Kernel Mode
- signals
- terminal handling
- character I/O system
- terminal drivers
- file system
- swapping block I/O system
- disk and tape drivers
- CPU scheduling
- page replacement
- demand paging
- virtual memory

Hardware
- terminal controllers
- terminals
- device controllers
- disks and tapes
- memory controllers
- physical memory
Virtualization: Execution Environments for Systems

Additional layers of protection and isolation can help further manage complexity

What is an Operating System, ... Really?

- Most Likely:
  - Memory Management
  - I/O Management
  - CPU Scheduling
  - Communications? (Does Email belong in OS?)
  - Multitasking/multiprogramming?

- What about?
  - File System?
  - Multimedia Support?
  - User Interface?
  - Internet Browser? 😊

- Is this only interesting to Academics??

Operating System Definition (Cont.)

- No universally accepted definition
- “Everything a vendor ships when you order an operating system” is good approximation
  - But varies wildly
- “The one program running at all times on the computer” is the kernel
  - Everything else is either a system program (ships with the operating system) or an application program

“In conclusion... Operating Systems:”

- Provide convenient abstractions to handle diverse hardware
  - Convenience, protection, reliability obtained in creating the illusion
- Coordinate resources and protect users from each other
  - Using a few critical hardware mechanisms
- Simplify application development by providing standard services
- Provide fault containment, fault tolerance, and fault recovery

- CS162 combines things from many other areas of computer science:
  - Languages, data structures, hardware, and algorithms