Introduction to Operating Systems and CS 162

Sam Kumar
CS 162: Operating Systems and System Programming
Lecture 1
https://cs162.eecs.berkeley.edu/

Read: A&D, Ch 1
Online Lecture Format

• Lectures will be held online on Zoom
• Webcasts will be uploaded to YouTube after lecture
• Ask questions on the Zoom chat
Introducing the CS 162 Team – Me

• Sam Kumar (samkumar@cs.berkeley.edu)
  • http://eecs.berkeley.edu/~samkumar
  • Office Hours: Mondays 3:30-5 PM, Fridays 9-11 AM

• PhD Student in the CS division
  • Research in systems, networking, and security
Introducing the CS 162 Team – Your TAs

William Hsu
whsu00@Berkeley.edu

Jonathan Shi
jhshi@berkeley.edu

Bobby Yan
bobbyy@berkeley.edu

Kevin Yu
kevin.j.yu@berkeley.edu
Today’s Objectives

• Introduce you to Operating Systems and their design
• Introduce the CS 162 instructional team and plan
• Establish expectations and logistics
• Get excited about how operating systems are essential in creating and advancing this “connected world” ...
Most Transformative Artifact of Human Civilization...

Map of the Internet in 1999
Running Systems at Internet Scale

Worldwide Internet Users

1969 1974 1990

ARPANet
RFC 675 TCP/IP
HTTP 0.9

Internet
WWW

0% 10% 20% 30% 40% 50% 60% 70%

0 500 1,000 1,500 2,000 2,500 3,000 3,500 4,000 4,500 5,000

% Population Million


0.9
WWW
Across Incredible Diversity

Bell’s Law: New computer class every 10 years

Computers Per Person

1:10^6

1:10^3

1:1

10^3:1

years

Mainframe

Mini

Workstation

PC

Laptop

PDA

Cell

Mote!

Number crunching, Data Storage, Massive Inet Services, ML, ...

Productivity, Interactive

Streaming from/to the physical world

The Internet of Things!
And Range of Timescales

Jeff Dean:  
“Numbers Everyone Should Know”

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 cache reference</td>
<td>0.5</td>
</tr>
<tr>
<td>Branch mispredict</td>
<td>5</td>
</tr>
<tr>
<td>L2 cache reference</td>
<td>7</td>
</tr>
<tr>
<td>Mutex lock/unlock</td>
<td>25</td>
</tr>
<tr>
<td>Main memory reference</td>
<td>100</td>
</tr>
<tr>
<td>Compress 1K bytes with Zippy</td>
<td>3,000</td>
</tr>
<tr>
<td>Send 2K bytes over 1 Gbps network</td>
<td>20,000</td>
</tr>
<tr>
<td>Read 1 MB sequentially from memory</td>
<td>250,000</td>
</tr>
<tr>
<td>Round trip within same datacenter</td>
<td>500,000</td>
</tr>
<tr>
<td>Disk seek</td>
<td>10,000,000</td>
</tr>
<tr>
<td>Read 1 MB sequentially from disk</td>
<td>20,000,000</td>
</tr>
<tr>
<td>Send packet CA-&gt;Netherlands-&gt;CA</td>
<td>150,000,000</td>
</tr>
</tbody>
</table>
Operating Systems 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, ...
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 a 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

Switchboard Operator

Computer Operators
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!
Hardware/Software Interface

What you learned in CS 61C – Machine Structures (and C)

The OS *abstracts* these hardware details from the application
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

Process: Execution environment with restricted rights provided by OS

Hardware

Operating System

Compiled Program
System Libs

Process: Execution environment with restricted rights provided by OS

Threads
Address Spaces
Files
Sockets
Compiled Program’s View of the World

Process: Execution environment with restricted rights provided by OS

- 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

ISA

Operating System

Compiled Program

System Libs

Threads
Address Spaces
Files
Sockets

ISA

Hardware

Process

Compiled Program’s View of the World

Compiled Program

System Libs

Threads
Address Spaces
Files
Sockets

Operating System

ISA

Hardware

Process

System Programmer’s View of the World

**Process: Execution environment with restricted rights provided by OS**

- 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)
  • ...

6/22/2020
For Example...
Operating System’s View of the World

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

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

- **Hardware**
  - Processor
  - ISA
  - Compiler
  - Networks
  - Storage
  - Memory
  - PgTbl & TLB
  - I/O Ctrlr
  - OS Mem

- **Compiled Programs**
  - Program 1
    - System Libs
  - Program 2
    - System Libs
Operating System’s View of the World

- OS translates from hardware interface to application interface
- OS provides each running program with its own process

Diagram:

- Processor
- Memory
- Storage
- Networks
- Operating System
- ISA
- Hardware
- Compiler
- Process 1
  - Threads
  - Address Spaces
  - Files
  - Sockets
- Process 2
  - Threads
  - Address Spaces
  - Files
  - Sockets
- System Libs
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

Process 1
- Compiler
- Process 1
  - Threads
  - Address Spaces
  - Files
  - Sockets
  - System Libs

Process 2
- Compiler
- Process 2
  - Threads
  - Address Spaces
  - Files
  - Sockets
  - System Libs

Operating System
- Processor
- Memory
- Storage
- Networks
- I/O Ctrlr

Hardware
- ISA

Compiler

6/22/2020
Kumar CS 162 at UC Berkeley, Summer 2020
OS Basics: Switching Processes

**Compiled Program 1**
- System Libs

**Compiled Program 2**
- System Libs

**Process 1**
- Threads
- Address Spaces
- Files
- Sockets

**Process 2**
- Threads
- Address Spaces
- Files
- Sockets

**Operating System**
- Processor
- Memory
- OS Mem

**Hardware**
- Compiler
- ISA

**Networks**
- Storage

**I/O**
- Ctrlr
- I/O Ctrlr
OS Basics: Switching Processes

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

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

Operating System
- Processor
- Memory
- OS Mem
- Storage
- Networks
- I/O Ctrlr

Compiler

ISA

Hardware
OS Basics: Switching Processes

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

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

Operating System
- Processor
- Memory
  - OS Mem
- Storage
- Networks
- I/O Ctrlr

Compiler

Hardware

ISA

6/22/2020
Kumar CS 162 at UC Berkeley, Summer 2020
OS Basics: Switching Processes

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

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

Operating System
- Threads
- Address Spaces
- Files
- Sockets

Compiler

Hardware

Processor

Memory

OS Mem

Storage

Networks

I/O Ctrlr

ISA

Processor

Compiler

Hardware
OS Basics: Protection

Compiled Program 1
System Libs

Process 1
Threads Address Spaces Files Sockets

Operating System

Compiled Program 2
System Libs

Process 2
Threads Address Spaces Files Sockets

Compiler

Processor

Memory

OS Basics: Protection

Compiled Program 1
System Libs

Process 1
Threads Address Spaces Files Sockets

Memory

I/O Ctrlr

Compiled Program 2
System Libs

Process 2
Threads Address Spaces Files Sockets

Storage

Networks

6/22/2020
Kumar CS 162 at UC Berkeley, Summer 2020
33
OS Basics: Protection

Compiled Program 1

System Libs

Process 1

Operating System

Compiler

Threads
Address Spaces
Files
Sockets

Process 2

Address Spaces
Files
Sockets

Compiled Program 2

System Libs

Segmentation fault (core dumped)

6/22/2020

Kumar CS 162 at UC Berkeley, Summer 2020
• **OS isolates** processes from each other

• **OS isolates** itself from other processes

• ... even though they are actually running on the same hardware!
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 provides common services in the form of I/O
OS Basics: Look and Feel

**Process:** Execution environment with restricted rights provided by OS

- **Thread**
- **Address Spaces**
- **Files**
- **Sockets**
- **Windows**

**Hardware**
- **ISA**
- **Processor**
- **Memory**
- **Storage**
- **Networks**
- **Displays**
- **I/O Ctrlr**

**Compiler**

**Compiled Program**
- **System Libs**

**Compiled Program**
- **Compiled Program**

**Process**

**Compiled Program**
- **Compiled Program**
OS Basics: Background Management

Process: Execution environment with restricted rights provided by OS

Operating System

Compiled Program
System Libs

Thread
Address Spaces
Files
Sockets
Windows

Hardware

Processor
Memory
Storage

 ISA

Compiler

Processor

PgTbl & TLB

OS Mem

I/O Ctrlr

ISA

Operating System

Network Manager

Power Manager

Network Manager

Battery
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: Hardware Support

• OS bottom line: support applications! The OS itself is incidental.
  • Ideally, OS should have very low performance overhead over the raw hardware

• At key points in the class, we will rely on support from the underlying hardware to implement OS abstractions efficiently:
  • Dual-mode operation, Interrupts, Traps, Precise exceptions, Memory Management Unit, Translation Lookaside Buffer, etc.

• Hardware support and OS design continue to co-evolve...
  • ... as hardware performance improves (e.g., faster storage/network), ...
  • ... and application requirements change.
  • What we study in this class is the result of decades of co-evolution!
Operating Systems...

• Provide consistent abstractions to apps, even on diverse hardware
  • File systems, Window Systems, Communications, ...
  • Processes, threads
  • VMs, containers
  • Naming systems

• Manage resources shared among multiple applications:
  • Memory, CPU, storage, ...

• Achieved by specific algorithms and techniques:
  • Scheduling
  • Concurrency
  • Transactions
  • Security

• Across immense scale – from 1’s to Billions
• Hopefully, at nearly the same performance as running on raw hardware!
CS 162 is a class about...

- The key systems abstractions that have emerged over time,
  - Processes, Threads, Events, Address Spaces, File Systems, Sockets, Transactions, Key-Value Stores, etc.

- The tradeoffs surrounding their design,

- Their efficient implementation,
  - Including the hardware support that makes them possible and practical

- And how to use them effectively.
Why take CS 162? Why learn about OS?

• Some of you will design and build parts of operating systems

• Many of you will create systems that use OS concepts
  • Whether you build hardware or software
  • The concepts and design patterns appear at many levels

• All of you will write programs that use OS abstractions
  • The better you understand their design and implementation, the better use you’ll make of them
Logistics and Administrivia
CS 162 over the Summer

• CS 162 is typically taught over a 16-week semester
• The summer offering of CS 162 is in Session C, which is 8 weeks long

• This class will run at twice the pace as a normal semester!
Syllabus, in a Nutshell

• OS Concepts and Abstractions (2 weeks)
  • Process, I/O, Networks, etc.

• Concurrency (2 weeks)
  • Threads, scheduling, synchronization, scalability, fairness

• Address Spaces (1 week)
  • Virtual memory, paging, address translation, protection, sharing

• File Systems (1 week)
  • I/O devices, files, storage, naming, caching, performance, transactions

• Distributed Systems (1 week)
  • Protocols, RPC, NFS, DHTs, Consistency, Scalability, Fault Tolerance

• Special Topics (1 week)
Textbook and Resources

• *Operating Systems, Principles and Practice, 2nd edition*, by Anderson and Dahlin

• 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

• Additional materials are on the “Readings” page on the course website
Useful Links

• Course Website:  
  https://cs162.eecs.berkeley.edu/

• Piazza for Q&A:  
  https://piazza.com/Berkeley/summer2020/cs162

• Anonymous Feedback Form:  
Learning by Doing

• Three Group Projects (Pintos in C)
  • User Programs
  • Scheduling
  • File Systems

• Individual Homeworks
  • Intro
  • Pintos Lists, Basic Threads
  • Build your own Shell
  • Sockets and Threads in an HTTP Server
  • Memory Mapping and Management
  • Key-Value Store (Go)
Group Projects

- Project teams have 4 members
  - Never 5, 3 requires serious justification

- Communication and cooperation will be essential
  - Design documents
  - You are encouraged to meet (virtually) with each other early and often

- Everyone should do work and have clear responsibilities
  - You will evaluate your teammates at the end of each project

- **Ultimately, every group member is responsible for understanding all aspects of the project!**
Project Components

1. Design Document and Design Review with TA
2. Code Checkpoint
3. Final Code
4. Peer Evaluations

• Regular **git push** so that TA sees your progress
Grading

• 40% exams
• 40% projects
• 15% homework
• 5% participation

• New this summer: CS 162 is graded on an absolute scale
  • The class will not be curved
  • Your final grade depends on how you do, not how your classmates do
Preparing Yourself for this Class

• Formal prerequisites: CS 61A, CS 61B, CS 61C, and CS 70

• We expect you to be familiar with:
  • Data structures: arrays, linked lists, binary trees, and hashing
  • Assembly language
  • C programming
  • How to debug C code, especially using GDB
  • CPU caches and memory hierarchy
  • Conceptual understanding of virtual memory
  • CPU pipelines and very basic digital logic design
  • Probability distributions (Normal & Exponential distributions)
Preparing Yourself for this Class

• The projects and homework assignments will require you to be very comfortable with programming and debugging in C
  • Pointers (including function pointers, void*)
  • Memory management (malloc, free, stack vs. heap)
  • Debugging with GDB

• Perhaps more important than formal prerequisites is experience working on large projects and debugging large C programs
  • You will be working on a larger, more sophisticated code base than anything you’ve likely seen in CS 61C
Preparing Yourself for this Class

• We will hold a C review session in the first week to re-acquaint you with the basics of C

• CS 161 will hold a C review session focusing on memory layout (also relevant to this class)

• See the “Course Info” page of the course website for additional resources to review the prerequisites

• We will provide an ungraded Quiz 0 to help you identify any topics you need to brush up on
Alternatives to CS 162

• CS 61C: Great Ideas of Computer Architecture (Machine Structures)
  • Offered in Summer 2020! 18 seats still open.
  • Covers machine structures
  • C programming, assembly programming, computer architecture, digital logic

• CS 9E: Productive Use of the UNIX Environment
  • Not offered over the summer
  • Focuses on how to use UNIX-like operating systems
  • Scripting and utilities
Class is Entirely Online!

• Due to COVID-19, the class will be taught entirely online this term

• Lectures will be on Zoom (recordings will be available)

• Discussions will be on Zoom
  • For the first two weeks, attend any discussion section you like
  • Once project groups are finalized, you’ll be assigned to a discussion section
  • We will try to time discussion sections so that everyone can attend
  • Exceptions granted only for hard conflicts
Online Exams

• Exams will also be online

• New this summer: There will be three quizzes and a final exam
  • If lowest score is on a quiz, that quiz is “dropped”
  • If lowest score is on the final, the final counts less

• We will hold the optional Quiz 0 using our online platform this week
  • To help you get familiar with the online exam platform
  • And work out any technical hurdles on our end
Personal Integrity

• UC Berkeley Academic Honor Code

“As a member of the UC Berkeley community, I act with honesty, integrity, and respect for others.”

• https://teaching.berkeley.edu/berkeley-honor-code
Collaboration Policy

OK:
• Explaining a concept to someone else (for projects, other groups)
• Discussing algorithms/testing strategies with others (for projects, other groups)
• Helping debug someone else’s code (for projects, in another group)
• Searching online for generic algorithms (e.g., hash table)

Not OK:
• Sharing code or test cases with someone else (for projects, other groups)
• Copying OR reading someone else’s code or test cases (for projects, in another group)
• Copying OR reading code or test cases online or from prior years
• Collaborating with other people during exams (quizzes or final)
Late Policy

• Deadlines are at 11:59 PM PDT (Pacific Daylight Time)

• 2 slip days to use for individual homework assignments
• 2 slip days to use (as a group) for projects

• After all slip days are used, no credit will be given for late work
First Assignments

• Homework 0 is out, due this Thursday
  • Exercises to re-acquaint yourselves with C and GDB
  • Also explores Executables

• **Optional** Quiz 0 is out
  • Will be held in online format this week

• “Project 0” is out this week (worth one point)
  • *Individual assignment, but open collaboration – for this assignment only!*
  • Opportunity to meet other students to form groups
  • Get familiar with Pintos, the OS used for class projects
What makes Operating Systems so Exciting and Challenging?

1. Evolving hardware
2. Managing complexity
What makes Operating Systems so Exciting and Challenging?

1. Evolving hardware
2. Managing complexity
Gordon Moore (co-founder of Intel) predicted in 1965 that the transistor density of semiconductor chips would double roughly every 18 months.

Called “Moore’s Law”

Microprocessors have become smaller, denser, and more powerful.
But then Moore’s Law Ended...

- 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
New Challenge: Slowdown in Joy’s Law

- This has caused a shift toward many-core chips
- Parallelism must be exploited at all levels!

Diagram taken from David Patterson’s Turing Lecture
The World is Parallel: Intel Skylake (2017)

- Up to 28 Cores, 58 Threads
  - 694 mm² die size (estimated)
- Many different instructions
  - Security, Graphics
- Caches on chip:
  - L2: 28 MiB
  - Shared L3: 38.5 MiB (non-inclusive)
  - Directory-based cache coherence
- Network:
  - On-chip Mesh Interconnect
  - Fast off-chip network directly supports 8 chips connected
- DRAM/chips
  - Up to 1.5 TiB
  - DDR4 memory
Storage Capacity

![Graph showing the increase in storage capacity over time. The graph illustrates the exponential growth in capacity, with distinct inflection points indicating technological shifts. The x-axis represents the year, ranging from 1980 to 2015, while the y-axis shows the capacity in GB. The graph highlights the rapid increase in capacity, especially after 2010, with marked improvements in SSD and 7.2K drives.]
Society is Increasingly Connected...
Network Capacity

Not Only PCs/Servers 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...
Societal Scale Information Systems

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

MEMS for Sensor Nets

Scalable, Reliable, Secure Services
- Databases
- Information Collection
- Remote Storage
- Online Games
- Commerce
...
What makes Operating Systems so Exciting and Challenging?

1. Evolving hardware
2. Managing complexity
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 well are they managed.
Hardware Functionality comes with Complexity

Proc
Caches
Memory
Busses

I/O Devices:
Controllers
Disks
Displays
Keyboards

Intel Skylake-X
I/O Configuration
Software Functionality comes with Complexity

Software is becoming more complex with time!

From MIT’s 6.033 course
Complexity Example: Mars Pathfinder

• Lots of types of hardware: Radios, science experiments, batteries, solar panels, motors and robotic equipment

• Unique reliability requirements
  • No reset button, must reboot automatically when necessary
  • Must always be able to receive commands from Earth

• Remote debugging
  • Software will crash, how is that handled?
  • Need to debug from Earth

• Time-critical functions
  • Stop before hitting a Mars boulder
  • Earth communications
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)
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
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
### UNIX System Structure

<table>
<thead>
<tr>
<th>Mode</th>
<th>Applications</th>
<th>Standard Libs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>User</strong></td>
<td>(the users)</td>
<td>shells and commands</td>
</tr>
<tr>
<td><strong>Mode</strong></td>
<td></td>
<td>compilers and interpreters</td>
</tr>
<tr>
<td><strong>Kernel</strong></td>
<td></td>
<td>system libraries</td>
</tr>
<tr>
<td><strong>Mode</strong></td>
<td></td>
<td><strong>system-call interface to the kernel</strong></td>
</tr>
<tr>
<td><strong>Kernel</strong></td>
<td></td>
<td><strong>signals terminal handling</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>character I/O system</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>terminal drivers</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>file system</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>swapping block I/O system</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>disk and tape drivers</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>CPU scheduling</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>page replacement</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>demand paging</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>virtual memory</strong></td>
</tr>
<tr>
<td><strong>Hardware</strong></td>
<td></td>
<td><strong>kernel interface to the hardware</strong></td>
</tr>
<tr>
<td><strong>Hardware</strong></td>
<td></td>
<td><strong>terminal controllers terminals</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>device controllers disks and tapes</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>memory controllers physical memory</strong></td>
</tr>
<tr>
<td><strong>Hardware</strong></td>
<td></td>
<td><strong>terminal controllers terminals</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>device controllers disks and tapes</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>memory controllers physical memory</strong></td>
</tr>
</tbody>
</table>
Virtualization: Execution Environments for Systems

Additional layers of protection and isolation can help further manage complexity.
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