Introduction to the Course

- Purpose of course and relationships to other courses
- Why study operating systems?
- Major themes & lessons in this course

What Will CS 111 Do?

- Build on concepts from other courses
  - Data structures, programming languages, assembly language programming, network protocols, computer architectures, ...
- Prepare you for advanced courses
  - Data bases and distributed computing
  - Security, fault-tolerance, high availability
  - Computer system modeling, queueing theory
- Provide you with foundation concepts
  - Processes, threads, virtual address space, files
  - Capabilities, synchronization, leases, deadlock

Why Study Operating Systems?

- Few of you will actually build OSs
- But many of you will:
  - Set up, configure, manage computer systems
  - Write programs that exploit OS features
  - Work with complex, distributed, parallel software
  - Work with abstracted services and resources
- Many hard problems have been solved in OS context
  - Synchronization, security, integrity, protocols, distributed computing, dynamic resource management, ...
  - In this class, we study these problems and their solutions
  - These approaches can be applied to other areas

Why Are Operating Systems Interesting?

- They are extremely complex
  - But try to appear simple enough for everyone to use
- They are very demanding
  - They require vision, imagination, and insight
  - They must have elegance and generality
  - They demand meticulous attention to detail
- They are held to very high standards
  - Performance, correctness, robustness,
  - Scalability, extensibility, reusability
- They are the base we all work from

Recurring OS Themes

- View services as objects and operations
  - Behind every object there is a data structure
- Separate policy from mechanism
  - Policy determines what can/should be done
  - Mechanism implements basic operations to do it
  - Mechanisms shouldn't dictate or limit policies
  - Must be able to change policies without changing mechanisms
- Parallelism and asynchrony are powerful and necessary
  - But dangerous when used carelessly

More Recurring Themes

- An interface specification is a contract
  - Specifies responsibilities of producers & consumers
  - Basis for product/release interoperability
- Interface vs. implementation
  - An implementation is not a specification
  - Many compliant implementations are possible
  - Inappropriate dependencies cause problems
- Modularity and functional encapsulation
  - Complexity hiding and appropriate abstraction

What Is An Operating System?

- Many possible definitions
- One is:
  - It is low level software . . .
  - That provides better abstractions of hardware below it
  - To allow easy, safe, fair use and sharing of those resources

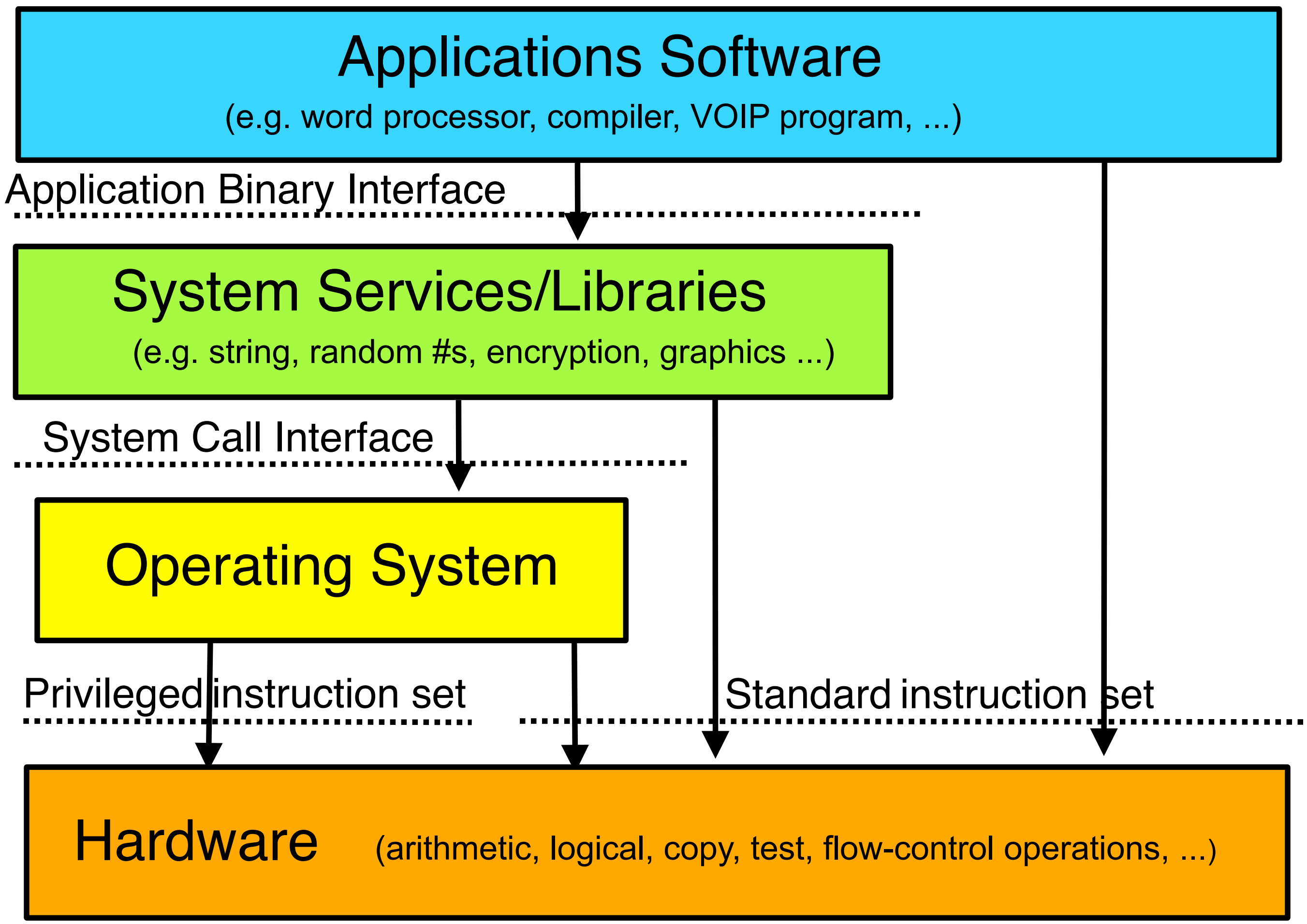
What Does an OS Do?

- It manages hardware for programs
  - Allocates hardware and manages its use
  - Enforces controlled sharing (and privacy)
  - Oversees execution and handles problems
- It abstracts the hardware
  - Makes it easier to use and improves s/w portability
  - Optimizes performance
- It provides new abstractions for applications
  - Powerful features beyond the bare hardware

What Does An OS Look Like?

- A set of management & abstraction services
  - Invisible, they happen behind the scenes
- Applications see objects and their services
  - CPU supports data-types and operations
    - Bytes, shorts, longs, floats, pointers, ...
    - Add, subtract, copy, compare, indirection, ...
  - So does an operating system, but at a higher level
    - Files, processes, threads, devices, ports, ...
    - Create, destroy, read, write, signal, ...
- An OS extends a computer
  - Creating a much richer virtual computing platform
    - Supporting richer objects, more powerful operations

Where Does the OS Fit In?



What's Special About the OS?

- It is always in control of the hardware
  - Automatically loaded when the machine boots
  - First software to have access to hardware
  - Continues running while apps come & go
- It alone has <u>complete access</u> to hardware
  - Privileged instruction set, all of memory & I/O
- It mediates applications' access to hardware
  - Block, permit, or modify application requests
- It is trusted
  - To store and manage critical data
  - To always act in good faith
- If the OS crashes, it takes everything else with it
  - So it better not crash . . .

What Functionality Is In the OS?

- As much as necessary, as little as possible
  - OS code is very expensive to develop and maintain
- Functionality must be in the OS if it ...
  - Requires the use of privileged instructions
  - Requires the manipulation of OS data structures
  - Must maintain security, trust, or resource integrity
- Functions should be in libraries if they ...
  - Are a service commonly needed by applications
  - Do not actually have to be implemented inside OS
- But there is also the performance excuse
  - Some things may be faster if done in the OS

The OS and Speed

- One reason operating systems get big is based on speed
- It’s faster to offer a service in the OS than outside it
  - If it involves processes communicating, working at app level requires scheduling and swapping them
  - The OS has direct access to many pieces of state and system services
  - The OS can make direct use of privileged instructions
- Thus, there’s a push to move services with strong performance requirements down to the OS

The OS and Abstraction

- One major function of an OS is to offer abstract versions of resources
  - As opposed to actual physical resources
- Essentially, the OS implements the abstract resources using the physical resources
  - E.g., processes (an abstraction) are implemented using the CPU and RAM (physical resources)
  - And files (an abstraction) are implemented using disks (a physical resource)

Why Abstract Resources?

- The abstractions are typically simpler and better suited for programmers and users
  - Easier to use than the original resources
    - E.g., don't need to worry about keeping track of disk interrupts
  - Compartmentalize/encapsulate complexity
    - E.g., need not be concerned about what other executing code is doing and how to stay out of its way
  - Eliminate behavior that is irrelevant to user
    - E.g., hide the sectors and tracks of the disk
  - Create more convenient behavior
    - E.g., make it look like you have the network interface entirely for your own use

Common Types of OS Resources

- Serially reusable resources
- Partitionable resources
- Sharable resources

Serially Reusable Resources

- Used by multiple clients, but only one at a time
  - Time multiplexing
- Require access control to ensure exclusive use
- Require graceful transitions from one user to the next
  - A switch that totally hides the fact that the resource used to belong to someone else
- Examples: printers, bathroom stalls

Partitionable Resources

- Divided into disjoint pieces for multiple clients
  – Spatial multiplexing
- Needs access control to ensure:
  – Containment: *you cannot access resources outside of your partition*
  – Privacy: *nobody else can access resources in your partition*
- Examples: disk space, dormitory rooms

Shareable Resources

- Usable by multiple concurrent clients
  - Clients do not have to "wait" for access to resource
  - Clients don't "own" a particular subset of resource
- May involve (effectively) limitless resources
  - Air in a room, shared by occupants
  - Copy of the operating system, shared by processes
- May involve under-the-covers multiplexing
  - Cell-phone channel (time and frequency multiplexed)
  - Shared network interface (time multiplexed)

General OS Trends

- They have grown larger and more sophisticated
- Their role has fundamentally changed
  - From shepherding the use of the hardware
  - To shielding the applications from the hardware
  - To providing powerful application computing platform
- They still sit between applications and hardware
- Best understood through services they provide
  - Capabilities they add
  - Applications they enable
  - Problems they eliminate

Another Important OS Trend

- Convergence
  - There are a handful of widely used OSs
  - New ones come along very rarely
- OSs in the same family (e.g., Windows or Linux) are used for vastly different purposes
  - Making things challenging for the OS designer
- Most OSs are based on pretty old models
  - Linux comes from Unix (1970s vintage)
  - Windows from the early 1980s

A Resulting OS Challenge

- We are basing the OS we use today on an architecture designed 30-40 years ago
- We can make some changes in the architecture
- But not too many
  - Due to compatibility
  - And fundamental characteristics of the architecture
- Requires OS designers and builders to shoehorn what's needed today into what made sense yesterday