Part 1: Introduction to OS

- Role of an OS
  - One layer in a computer system architecture
  - Top-down view: convenient user interface
  - Bottom-up view: efficient resource manager
  - Design an OS

- OS Architecture
- OS History and Features
- Types of OS

From Computer System to OS

- 计算机系统的构成：硬件，软件
- 观察OS的两个视角：
  - 从硬件的角度：资源的管理者 ⇒ 统一于：
  - 从软件的角度：服务的提供者 运行就是服务

Computer Hardware

- Computer System
What does an Operating System do?

- Coordinator and Traffic Cop (An OS is Similar to a government):
  - Manages all resources
  - Settles conflicting requests for resources
  - Prevent errors and improper use of the computer

- Facilitator (“useful” abstractions):
  - Provides facilities/services that everyone needs
  - Standard Libraries, Windowing systems
  - Make application programming easier, faster, less error-prone

The Operating System as an Extended Machine

- Operating systems turn ugly hardware into beautiful abstractions

Virtual Machines

- Software emulation of an abstract machine
  - Make it look like hardware has features you want
  - Programs from one hardware & OS on another one
- Programming simplicity
  - Each process thinks it has all memory/CPU time
  - Each process thinks it owns all devices
  - Different Devices appear to have same interface
  - Device Interfaces more powerful than raw hardware
    - Bitmapped display ⇒ windowing system
    - Ethernet card ⇒ reliable, ordered, networking (TCP/IP)
- Fault Isolation
  - Processes unable to directly impact other processes
  - Bugs cannot crash whole machine
- Protection and Portability
  - Java interface safe and stable across many platforms
Principles of Virtualization

Summary: OS’s Functionalities
- **Top-down (user) view**
  - the O/S is a convenient application interface
    - the O/S hides the messy details which must be performed
    - the O/S presents user with a virtual machine easier to use
- **Bottom-up (hardware) view**
  - the O/S performs efficient resource usage and management
    - time multiplexing: each program gets to use a resource
    - space multiplexing: each program gets part of a resource
- **Software layer view**
  - the O/S is an evolvable and scalable software
    - the O/S permits effective development and introduction of new system functions without interfering with service

Role of an OS
- **The Stallings “stairs” view**
  - From: “Operating Systems: Internals and Design Principles” by Stallings

Operating System Definition
- **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
OS as a Program

- 传统的控制系统中，控制者与控制对象是分离的（比如供热系统与自动调温器）
- OS的控制是不同的
  - CPU只有一个
  - 只能有一个程序被CPU执行
  - OS也是一个程序

Increasing Software Complexity

- A programmer can produce only 1000 lines of debugged code per year on large projects
- Large project design
  - 1/3 Planning
  - 1/6 Coding
  - 1/4 Module testing
  - 1/4 System testing
- People and time not interchangeable
  - Work cannot be fully parallelized
  - Work must be partitioned into large numbers of modules
  - Debugging is highly sequential
- No Silver Bullet:
  - better high-level languages, artificial intelligence, expert systems...

为什么设计一个“好的”OS是很困难的？

- 根据摩尔定律，硬件每10年改进100倍，但现在的操作系统在某些方面（比如可靠性）比70年代的Unix版本还糟糕
- 为什么会出现这种状况呢？
  - OS的设计者对于需要什么必须非常清楚，但什么是“好的”OS并不明确
  - OS所支持的app一直在以无法预期的方式快速变化
  - OS无法有效地解耦成独立的模块。航空母舰比OS更复杂，但是航母能更好地分成相互隔离的部分
  - 没有人能理解500万行的代码，结果无法充分优化也就不奇怪了
  - ......
Part 1: Introduction to OS

- Role of an OS
- OS Architecture
  - Monolithic structure
  - Layered structure
  - Microkernel structure
- OS History and Features
- Types of OS

Why software architecture?

- Operating systems have become huge complex beasts
- With code size come all the problems
  - OS are chronically late in being delivered (new or upgrades)
  - OS have latent bugs that pop up and must be quickly fixed
  - performance is often not what was expected
  - it has proven impossible to deploy an OS that is not vulnerable to security attacks
- Hence the critical need for a well-engineered software architecture
  - layers and/or modules with clean, minimal interfaces
  - the goal is that one part can be easily changed (fixed, upgraded, expanded) without impacting the other parts

Why software architecture?

- Well-defined interfaces allow part replacement without impacting the surroundings

Monolithic Structure

- “The Big Mess”: a collection of procedures that can call any of the other procedures whenever they need to
- Bad examples: MS-DOS, original UNIX
Layered Structure

- Monolithic operating systems
  - no one had experience in building truly large software systems
  - the problems caused by mutual dependence and interaction were grossly underestimated
  - such lack of structure became unsustainable as OS grew
- Enter hierarchical layers and information abstraction
  - each layer is implemented exclusively using operations provided by lower layers
  - it does not need to know how they are implemented
  - hence, lower layers hide the existence of certain data structures, private operations and hardware from upper layers

Layered Structure

- Theoretical model of operating system design hierarchy

Layered Structure

- Major difficulty with layering
  - ... appropriately defining the various layers!
  - layering is only possible if all function dependencies can be sorted out into a Directed Acyclic Graph (DAG)
  - however there might be conflicts in the form of circular dependencies ("cycles")
Layered Structure

- Circular dependencies in an O/S organization
  - example: disk driver routines vs. CPU scheduler routines
- Other difficulty: efficiency
  - the more layers, the more indirections from function to function and the bigger the overhead in function calls
  - backlash against strict layering: return to fewer layers with more functionality

Microkernel Structure

- a microkernel is a reduced operating system core that contains only essential O/S functions
- the idea is to “defat” the kernel by moving up as much functionality as possible from the kernel into user space
- many services traditionally included in the O/S are now external subsystems running as user processes
  - device drivers
  - file systems
  - virtual memory manager
  - windowing system
  - security services, etc.

Microkernel Structure

- Benefits of the microkernel approach
  - extensibility—it is easier to extend a microkernel-based O/S as new services are added in user space, not in the kernel
  - portability—it is easier to port to a new CPU, as changes are needed only in the microkernel, not in the other services
  - reliability & security—much less code is running in kernel mode; failures in user-space services don’t affect kernel space
- Detriments of the microkernel approach
  - again, performance overhead due to communication from user space to kernel space
  - not always realistic: some functions (I/O) must remain in kernel space, forcing a separation between “policy” and “mechanism”

Part 1: Introduction to OS

- Role of an OS
- OS Architecture
- OS History and Features
  - Serial processing
  - batch systems
  - Multiprogramming
  - Time-sharing systems
- Types of OS
History of Operating Systems

Generations

- 1945–1955
  - Vacuum Tubes
  - Serial Processing

- 1955–1965
  - Transistors
  - Batch Systems

- 1965–1980
  - ICs
  - Multiprogramming
  - Time-sharing systems

- 1980–Present
  - LSIs & Personal Computers

Serial Processing

First generation: 1945–1955
- room full of armoires: mechanical relays, then vacuum tubes

Human operator-programmer-user
Operating systems were unheard of
Programs directly access the hardware, one at a time

Batch Systems

Second generation: 1955–1965
- advent of transistors and printed circuits
Batch Systems

Memory layout for a resident monitor

Multiprogramming

- Third generation: 1965-1980
  - first major use of small-scale Integrated Circuits (ICs)

Problem: despite batching, a lot of CPU time is still wasted waiting for I/O instructions to complete
  - I/O devices much slower than processor (especially tapes!)
Multiprogramming

- Uni-programming
  - CPU work
    - User program
    - Monitor program
    - I/O operation
  - User program

- Multiprogramming
  - Program A
  - Program B
  - CPU
  - Disk
  - Input
  - Output
  - Input

Multiprogramming

Summary: serial, batched uni-, and multiprogramming

Time-sharing Systems

- There was a need for multiple-user interactivity
  - Multiprogramming alone does not give any guarantee that a program will run in a timely manner
  - Each user wants to see their program running as if it was the only program in the computer

- In the original multiprogramming systems
  - Tasks kept running until they performed an operation that required waiting for an external event such as I/O
  - Multiprogramming was designed to maximize CPU usage

- In time-sharing systems
  - Running tasks are required to relinquish the CPU on a regular basis through hardware interrupts (timer)
  - Time-sharing is designed to minimize response time and allow several programs to execute apparently simultaneously
  - Time sharing is a logical extension of multiprogramming for handling multiple interactive jobs among multiple users

Personal Computers

- Fourth generation: 1980-Present
  - Large Scale Integration (LSI) makes personal computing real
  - From multiple users back to a single user
    - Preemptive multitasking was developed in the 1960’s to share big and costly mainframe computers among multiple users
    - Since then, single-user interactive computing has become possible on dedicated personal computers
  - Resource sharing not critical anymore, yet multitasking still a central feature of modern PC operating systems
    - A single-tasking environment is tedious: one must close the drawing application before opening the word processor, etc.
    - In time-sharing systems, the primary goal of PC systems is to maximize user convenience and responsiveness
Part 1: Introduction to OS

- Role of an OS
- OS Architecture
- OS History and Features
- Types of OS
  - Mainframe systems
  - Desktop & laptop systems
  - Parallel systems
  - Real-time systems

In Conclusion…

- OS as illusionist:
  - Make hardware limitations go away
  - Provide illusion of dedicated machine with infinite memory and infinite processors
- OS as government:
  - Protect users from each other
  - Allocate resources efficiently and fairly
- OS as history teacher
  - Learn from past
  - Adapt as hardware tradeoffs change
- OS as complex system:
  - Constant tension between simplicity and functionality or performance

After the class…

- Reading:
  - 教材第一章: 硬件知识回顾 pp.7-45 (8th Edition)
  - 教材第二章: 操作系统概述 pp.46-104 (8th Edition)
- Homework:
  - P37: Problems 1.3, 1.4, 1.7, 1.8, 1.9 (8th Edition)
  - P103: Problems 2.1, 2.2, 2.3, 2.4, 2.6 (8th Edition)
- Reminder:
  - Finishing reading the textbook in 2 weeks!