
CS2106

Introduction to OS

Lecture 1 Introduction

Overview

■ Operating Systems basic concepts:

- ❑ What is OS?
- ❑ Brief History
 - Motivation for OS
- ❑ Overview of Modern OSes

■ Operating System Structures

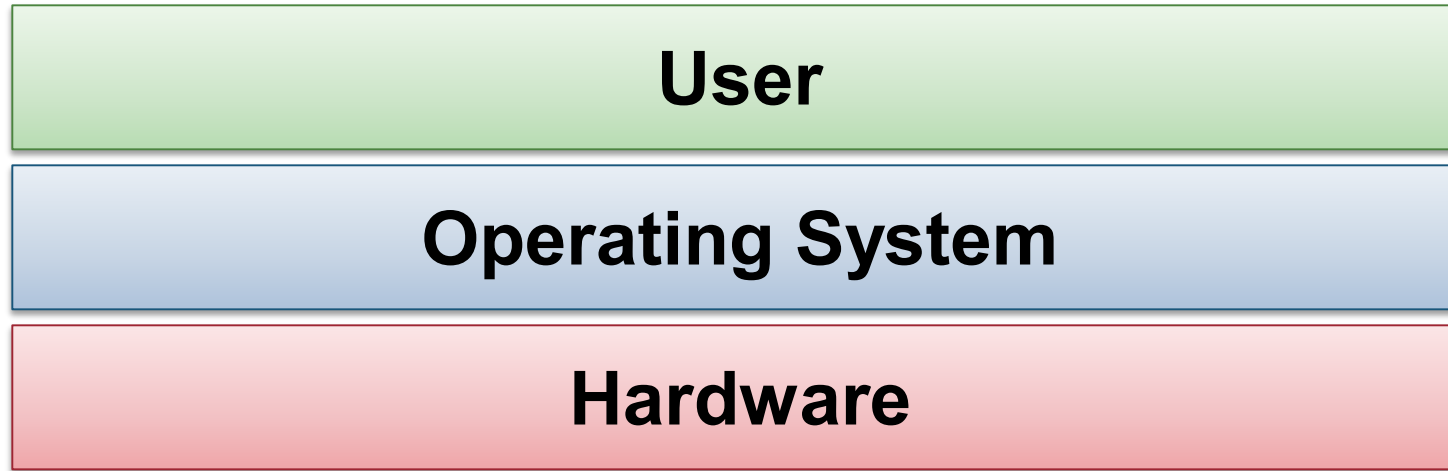
- ❑ OS components
- ❑ Types of kernel

■ Virtual Machines

What is OS?

- Incorrect/Incomplete definition:
 - ❑ It is the desktop when you boot up your PC
 - ❑ The "thing" that stores your games
 - ❑ Windows! (or Mac!) (or Linux!)
- One simple definition:
 - ❑ A **program** that acts as an **intermediary** between a **computer user** and the **computer hardware**
 - ❑ **Wikipedia:** An operating system (OS) is system software that manages computer hardware, software resources, and provides common services for computer programs.

Illustration: What is an OS?



- A simplified view:
 - Will be refined as we move along
- The most general version:
 - Hardware (not only computer!)
 - User (can be application programs or actual person!)

Example of Common OS

- On Computer:

- ❑ Windows 10/8/XP
- ❑ Mac OS X
- ❑ Linux distros: Ubuntu, Redhat, Debian, Fedora, CentOS
- ❑ Solaris, FreeBSD

- On Smartphone:

- ❑ iOS, Android, Windows Mobile

- Other hardware with OS:

- ❑ Game console: PS4, Xbox, Nintendo Switch, ...
- ❑ Home appliance: Blu-ray/DVD Player, Mio Box, ...

To invent the future, you must understand the past

BRIEF HISTORY OF OS

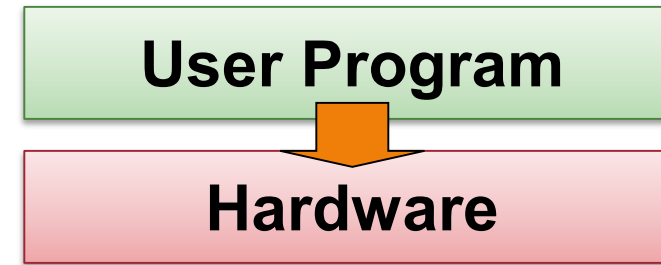
Brief History of OS

- Essentially, OS evolves with:
 - ❑ Computer hardware
 - ❑ User application and usage pattern
- The "first" computers:
 - ❑ Electronic Numerical Integrator And Computer (**ENIAC**)
 - 1945
 - Program controlled by cables and switches
 - ❑ Harvard Mark I:
 - 1944
 - Program controlled by punched paper tape

OS for the first computers

- OS Type:

- ❑ **NO OS**



- Programs directly interact with hardware

- ❑ Reprogram by changing **physical** configuration of hardware

- **Advantage:**

- ❑ Minimal overhead

- **Disadvantage:**

- ❑ Not portable

- ❑ Inefficient use of computer!

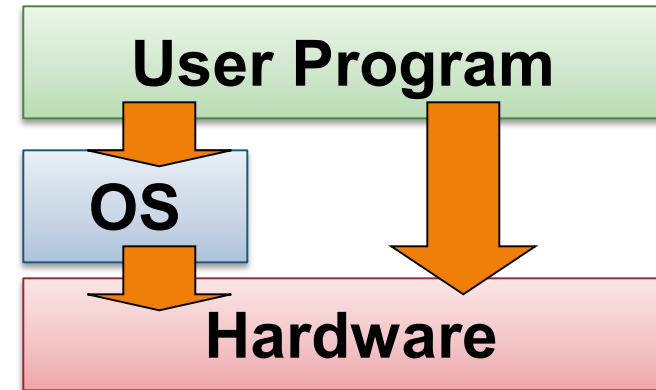
Mainframes: The "Big Iron"

- Commonly used by large corporations in 60s, 70s
- Common features:
 - ❑ No interactive interface
 - ❑ Accept programs in the form of:
 - Paper tape, magnetic tape, punch card
 - ❑ Support batch processing only
 - ❑ Very costly
 - Usually "rented" instead of owned
- Example:
 - ❑ IBM 360
 - Cost 5 billion US dollars in 1964 to develop
 - Cost 130k US dollar in 1965 to buy

OS for Mainframes

- OS Type:

- ❑ Batch OS



- Batch OS:

- ❑ Execute user program (a.k.a **job**) one at a time
 - Load job from media, execute, collect result

- User Job:

- ❑ Still interact with hardware directly
 - ❑ With additional information for the OS
 - Resource required
 - Job specification

OS for Mainframes: Improvements

- Simple batch processing is inefficient:
 - ❑ CPU idle when perform I/O
- One possible Improvements:
 - ❑ Multiprogramming:
 - loads multiple jobs and runs other jobs when I/O needs to be done
 - Overlaps computation with I/O
- Another development of OS during this period (70s):
 - ❑ Time-Sharing OS

Time-Sharing OS

- Features:

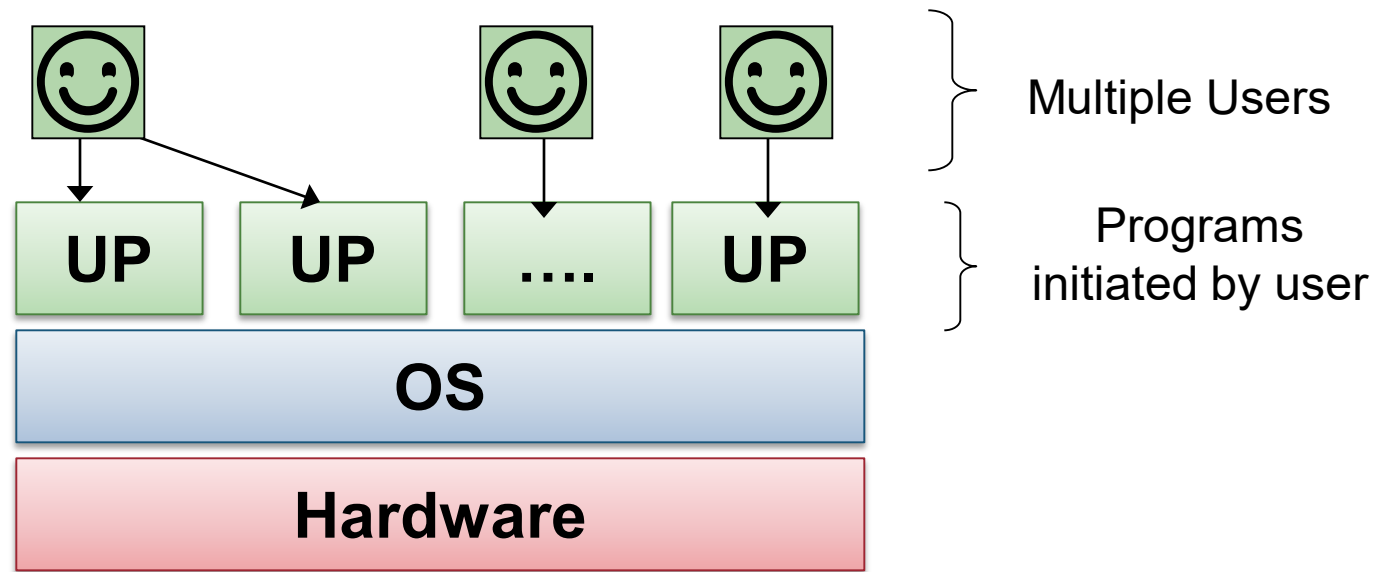
- ❑ Allow multiple users to interact with machine using terminals (***teletypes***)
- ❑ User job scheduling
 - Illusion of Concurrency
- ❑ Memory management

- Famous Examples:

- ❑ CTSS developed at MIT 1960s
- ❑ Multics (1970s)
 - Considered as the *parent of Unix*
- ❑ Pushed the state of art in virtual memory, security

- Similar to Unix servers today but more primitive

Time-sharing OS: Illustration



- OS manages the sharing of:
 - ❑ CPU time, memory and storage
- **Virtualization** of hardware:
 - ❑ Each program executes **as if** it has all the resources to itself

Minicomputer and Unix

- Minicomputer follows the mainframe:
 - A "mini" version of mainframe:
 - Smaller and cheaper
 - Example:
 - Digital Equipment Corp (DEC) PDP-11
- Famous OS:
 - Unix
 - Developed by AT&T employees, including Ken Thompson, Dennis Ritchie, Douglas McIlroy, and Joe Ossanna
 - Ken Thompson and Dennis Ritchie
 - Invented the C programming language as well!!

Personal Computer

- Apple II PC (1977):
 - ❑ First successfully produced mass home computer
 - ❑ Designed by Steve Wozniak (alone!)
- IBM PC (1981):
 - ❑ The first generic PC
 - ❑ PC becoming a collection of **commodity** hardware components
 - ❑ Leads to dominance of Microsoft OSes on PCs: MSDOS (1981) then Windows (1985)

OS on Personal Computer

- Machine (can be) dedicated to user, not timeshared between multiple users
 - Give rise to **personal OS**
- Several Models:
 - Windows model:
 - Single user at a time but possibly more than 1 user can access
 - Dedicated machine
 - Unix model:
 - One user at the workstation but other users can access remotely
 - General time sharing model

Why do we need OS?

MOTIVATIONS OF OS

Motivation for OS: Abstraction

- Large variation in hardware configurations
- Example (Hard disk):
 - Different capacity (500mb, 320gb, 1.5tb etc)
 - Different capabilities:
 - Rotation per minutes (RPM)
 - Access (read/write) speed
 - etc.
- However, hardware in the same category has well defined and common functionality
 - Example (Hard disk): Store and retrieve information

Motivation for OS: **Abstraction**

- Operating System serves as an **abstraction**:
 - Hide the different low level details
 - Present the common high level functionality to user
- The user can then perform essential tasks **through** operating system
 - no need to concern with low level details
- Provides:
 - Efficiency, programmability and portability

Motivation for OS: **Resource Allocator**

- Program execution requires multiple resources:
 - CPU, memory, I/O devices etc.
- For better utilization of resources, multiple programs should be allowed to execute simultaneously
- OS is a **resource allocator**
 - Manages all resources
 - CPU, Memory, Input/Output devices
 - Arbitrate potentially conflicting requests
 - for efficient and fair resource use

Motivation for OS: **Control Program**

- Program can misuse the computer:
 - Accidentally: due to coding bugs
 - Maliciously: virus, malware etc.
- Multiple users can share the computer:
 - Tricky to ensure separate user space
- OS is a **control program**
 - Controls execution of programs
 - Prevent errors and improper use of the computer
 - Provides security, isolation and protection

Motivation for OS: Summary

- Manage resources and coordination
 - process synchronization, resource sharing
- Simplify programming
 - abstraction of hardware, convenient services
- Enforce usage policies
- Security and protection
- User Program Portability:
 - Across different hardware
- Efficiency
 - Sophisticated implementations
 - Optimized for particular usage and hardware

The families of modern OS

OVERVIEW OF MODERN OS

Modern OS: Overview

PC



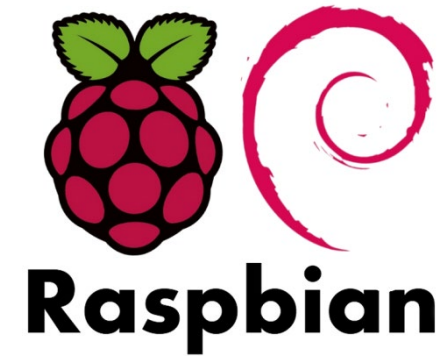
Mobile



Real-Time

freeRTOS

Embedded



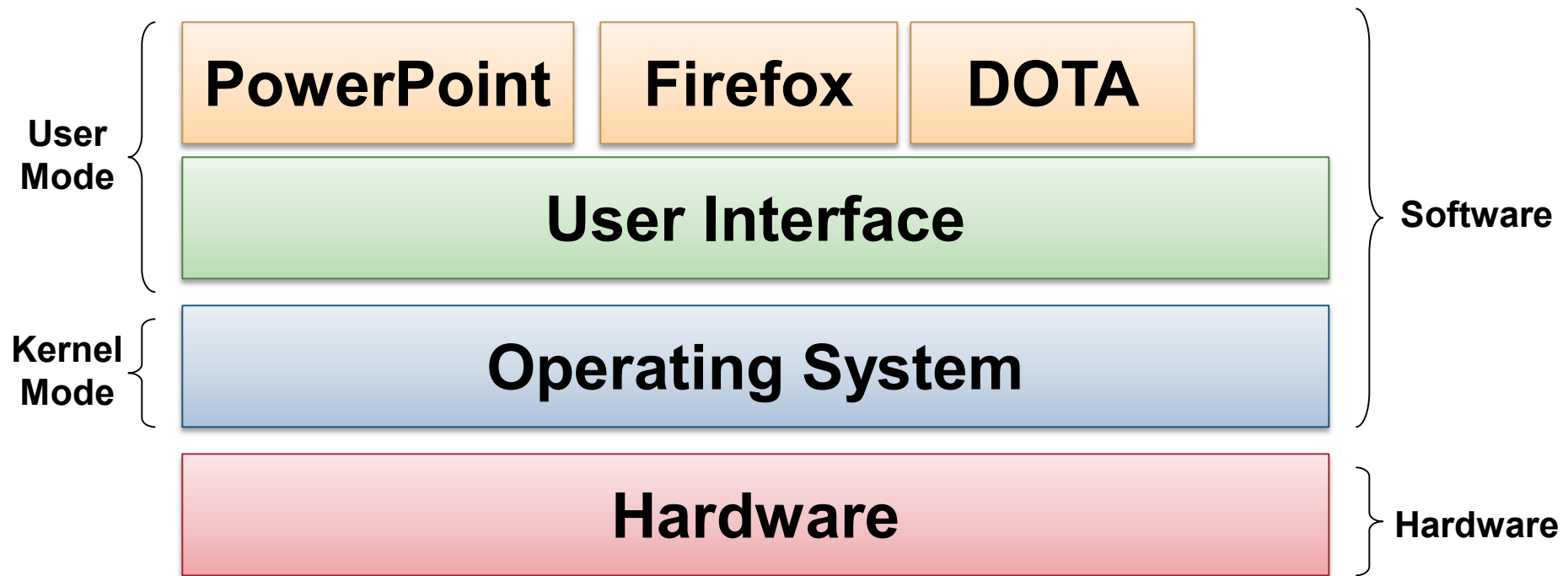
Common Architecture for OS

OS STRUCTURE

Operating System Structures

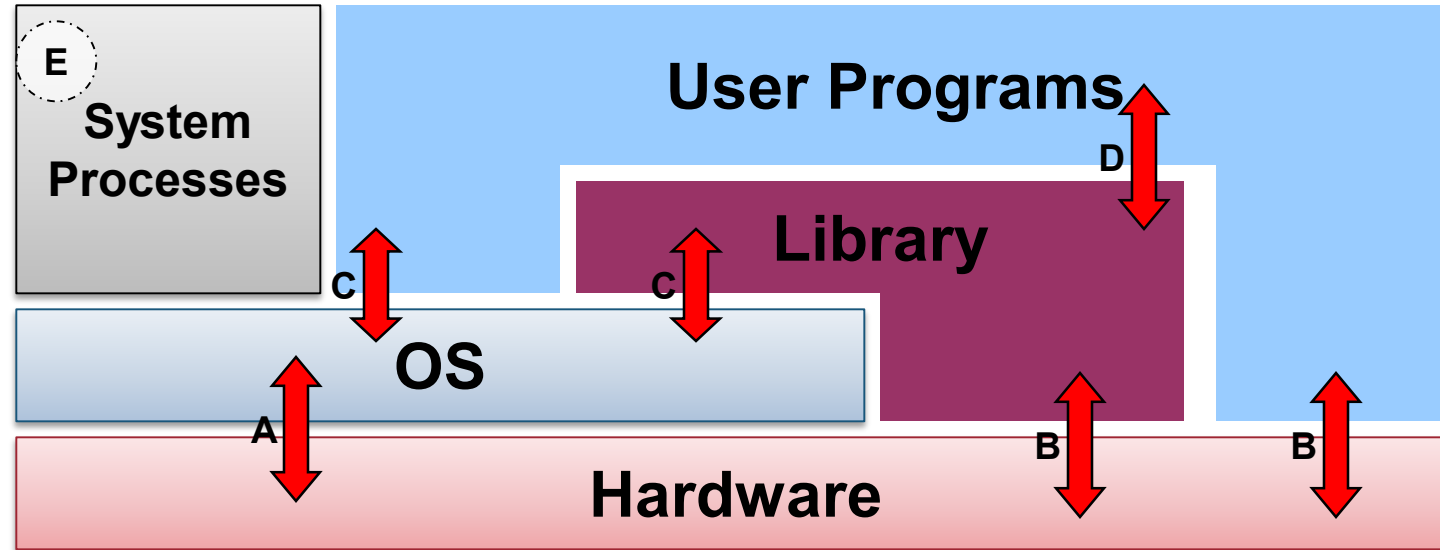
- We have identified the major capabilities of an OS
 - i.e. the *specification* of the OS
- Let us now consider:
 - The best way to provides these capabilities
 - i.e. the *implementations* of the OS
- Operating system structure:
 - *Organization* of the various components
 - Important factors:
 - Flexibility
 - Robustness
 - Maintainability
 - Performance

Illustration: High level view of OS



- Operating System is essentially a software
 - ❑ Runs in **kernel mode**: complete access to all hardware resources
- Other software executes in **user mode**
 - ❑ With limited (or controlled) access to hardware resources

Illustration: Generic OS Components



- **A**: OS executing machine instructions
- **B**: normal machine instructions executed (program/library code)
- **C**: calling OS using ***system call interface***
- **D**: user program calls library code
- **E**: system processes
 - Provide high level services, usually part of OS

OS as a Program

- OS is also known as the **kernel**
 - ❑ Just another program with some special features
 - Deals with hardware issues
 - Provides system call interface
 - Special code for interrupt handlers, device drivers
- Kernel code has to be different than normal programs:
 - ❑ no use of system call in kernel code
 - ❑ can't use normal libraries
 - ❑ no normal I/O
- Consider this:
 - ❑ Normal programs use OS: what does OS use? 😊

Implementing Operating System

■ Programming Language:

- ❑ Historically in assembly/machine code
- ❑ Now in **HLLs**:
 - Especially C/C++
- ❑ Heavily hardware architecture dependent

■ Common code organization:

1. Machine independent HLL
2. Machine dependent HLL
3. Machine dependent assembly code

■ Challenges:

- ❑ “No one else” to rely on for nice services
- ❑ Debugging is hard
- ❑ Complexity
- ❑ Enormous Codebase

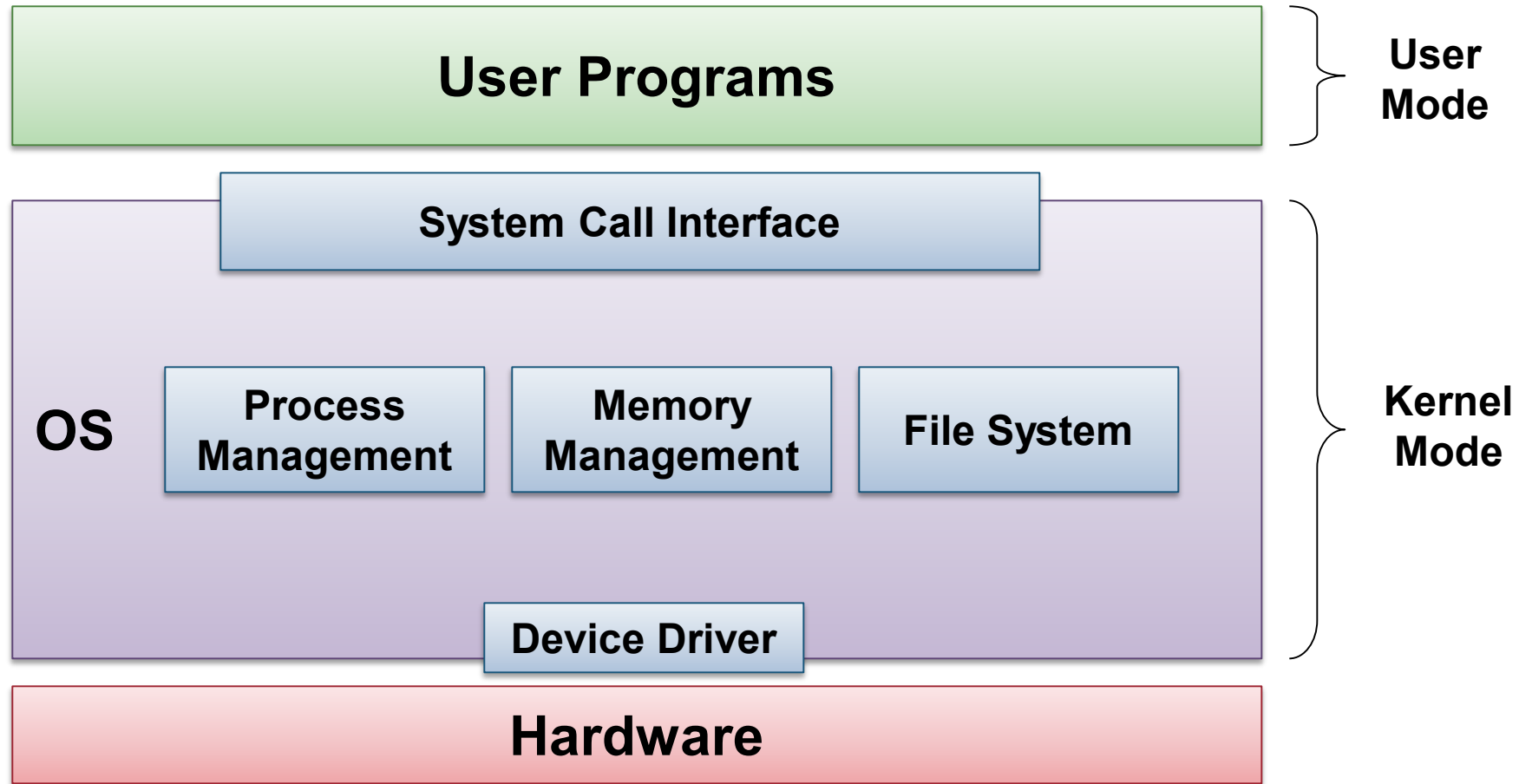
OS Structures

- Several ways to structure an OS:
 - ❑ **Monolithic**
 - ❑ **Microkernel**
 - ❑ Layered
 - ❑ Client-Server
 - ❑ Exokernel
 - ❑ Hybrid
 - ❑ etc.
- We will cover the first two in details:
 - ❑ They represent the whole range of possibilities
 - ❑ Most other approaches are variant or improvement

Monolithic OS

- **Kernel is:**
 - ❑ One **BIG** special program
 - Various services and components are integral part
 - ❑ Good SE principles are still possible with:
 - modularization
 - separation of interfaces and implementation
- This is the traditional approach taken by:
 - ❑ Most Unix variants, DOS, Windows 9x
- **Advantages:**
 - ❑ Well understood
 - ❑ Good performance
- **Disadvantages:**
 - ❑ Highly coupled components
 - ❑ Usually devolved into very complicated internal structure

Monolithic Kernel Illustration

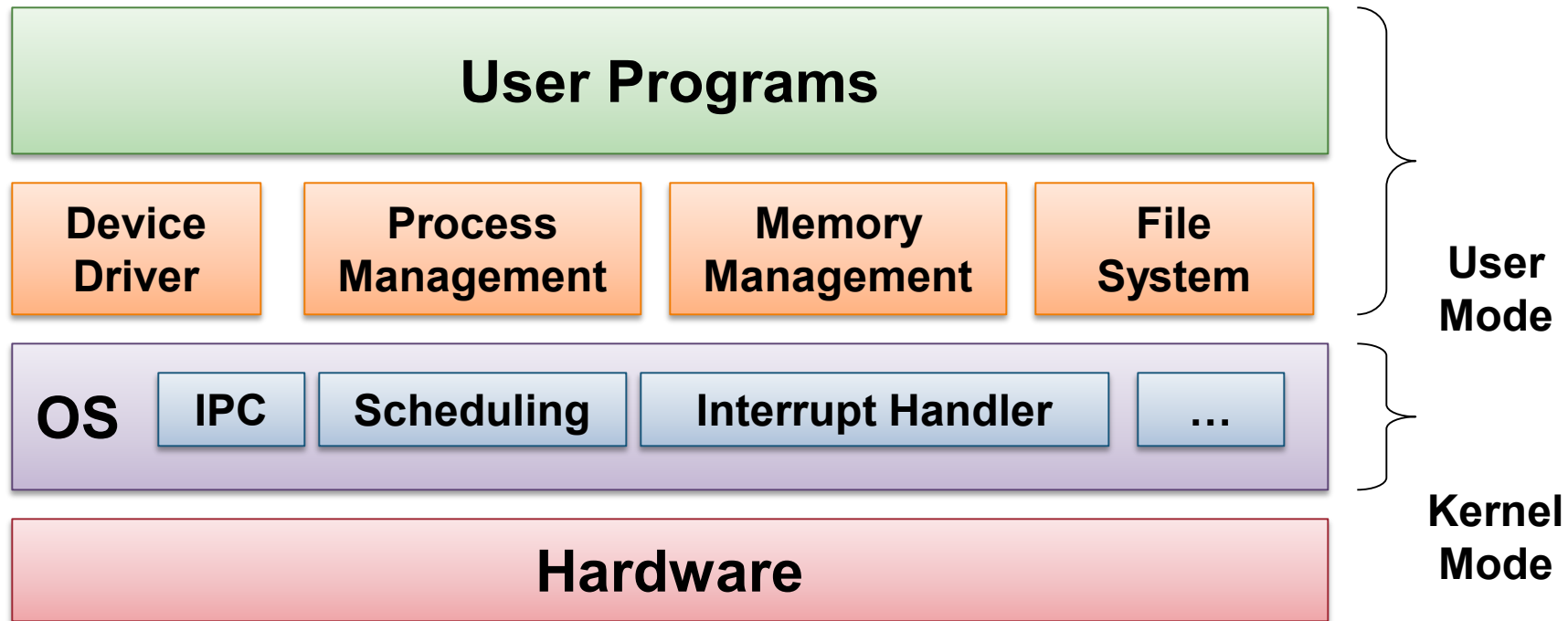


Generic Architecture of Monolithic OS Components

Microkernel OS

- Kernel is:
 - ❑ Very small and clean
 - ❑ Only provides basic and essential facilities:
 - Inter-Process Communication (IPC)
 - Address space management
 - Thread management
 - etc.
- Higher level OS services:
 - ❑ Built on top of the basic facilities
 - ❑ Run as server process **outside** of the kernel
 - ❑ Use IPC to communicate
- Advantages:
 - ❑ Kernel is generally more robust and more extendible
 - ❑ Better isolation and protection between kernel and high level services
- Disadvantages:
 - ❑ Lower Performance

Microkernel Components



Generic Architecture of Microkernel OS Components

Other Operating System Structure

■ Layered Systems:

- ❑ Generalization of monolithic system
- ❑ Organize the components into hierarchy of layers
 - Upper layers make use of the lower layers
 - Lowest layer is the hardware
 - Highest layer is the user interface

■ Client-Server Model

- ❑ Variation of microkernel
- ❑ Two classes of processes:
 - Client process request service from server process
 - Server Process built on top of the microkernel
 - Client and Server process can be on separate machine!

Ways of running OSes

VIRTUAL MACHINES

Motivation: Why Virtual Machines

- OS assumes total control of the hardware:
 - ❑ What if we want to run several OSes on the same hardware at the same time?
 - ❑ Cloud computing (IaaS)
- OS is hard to debug / monitor:
 - ❑ How do we observe the working of the OS?
 - ❑ How do we test a potentially destructive implementation?

Definition: Virtual Machine

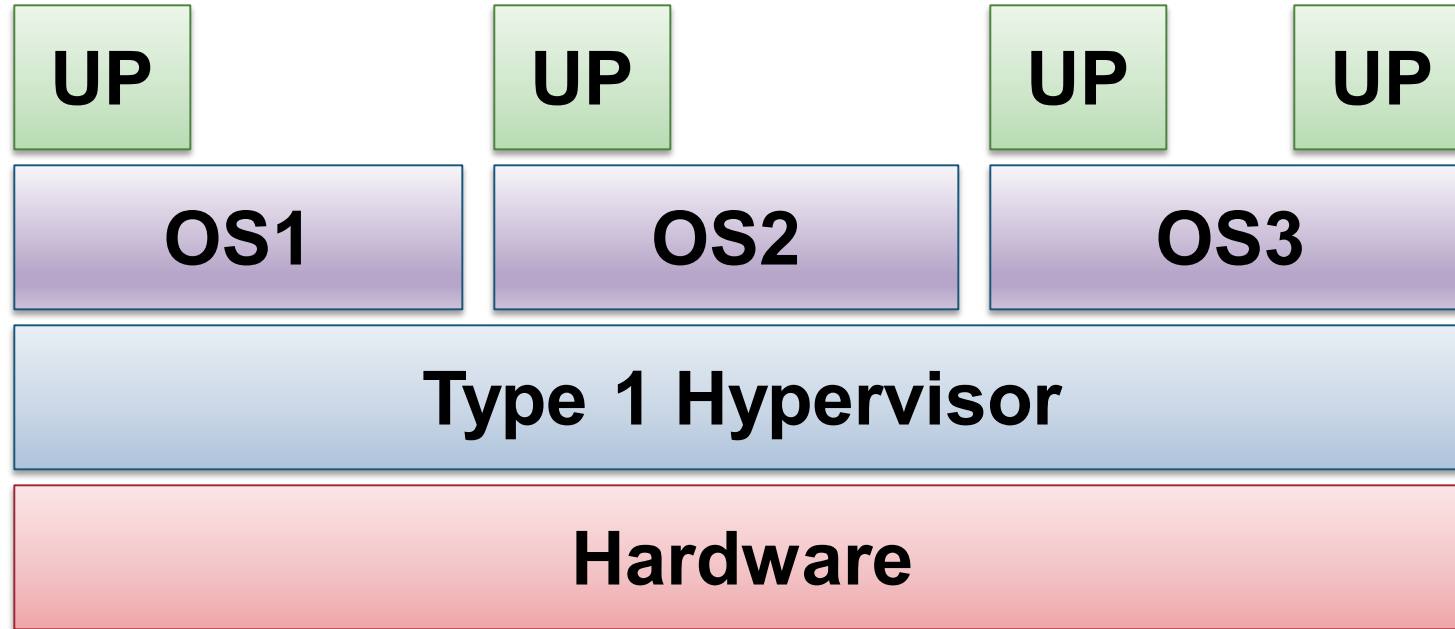
- Virtual Machine:

- A software emulation of hardware
- **Virtualization** of underlying hardware
 - Illusion of complete hardware to level above: memory, CPU, hard disk etc...
- Normal (primitive) operating system can then run on top of the virtual machine

- Created and managed by **Hypervisor**

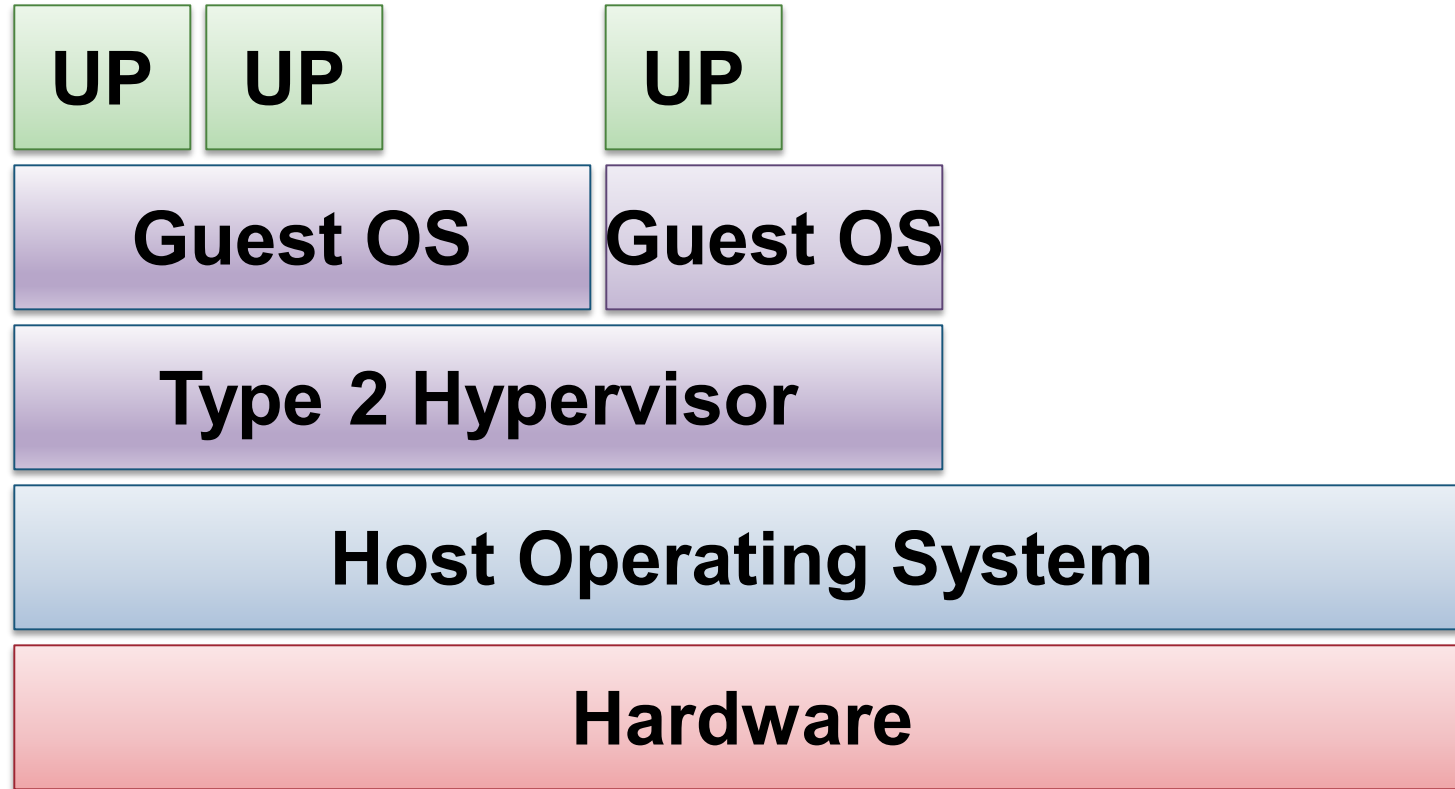
- Also known as **Virtual Machine Monitor (VMM)**
- Two classes of implementations shown next

Type 1 Hypervisor



- Type 1 hypervisor:
 - ❑ Provides individual *virtual machines* to guest OSes
 - ❑ eg. IBM VM/370

Type 2 Hypervisor



- Type 2 hypervisor OS
 - ❑ Runs in host OS
 - ❑ Guest OS runs inside Virtual Machine
 - ❑ e.g. VMware

Summary

- Definition of Operating System
- Roles of Operating System
- Common Operating System families
- Operating System structure

Reference

- Modern Operating System (4th Edition)
 - By Andrew S.Tanenbaum
 - Published by Pearson
- Operating System Concepts (8th Edition)
 - By Abraham Silberschatz, Peter Baer Galvin & Greg Gagne
 - Published by McGraw Hill