



Introduction to Operation Systems

2023-24 COMP3230B

Learning Outcome

- ILO 1 – [**Fundamentals**] discuss the **characteristics of different structures** of the Operating Systems (such as microkernel, layered, virtualization, etc.) and identify **the core functions** of the Operating Systems.

Contents

- Purpose and Functions of Operating Systems
- Core Services
- OS Architectures (Implementation design)

Reading & Reference

- Required Readings
 - Chapter 2, Introduction to Operating Systems, Operating Systems: Three Easy Pieces by Arpaci-Dusseau et. al
 - <http://pages.cs.wisc.edu/~remzi/OSTEP/intro.pdf>
 - Section 1.10 to 1.13 of Chapter 1 of Operating Systems, 3rd edition by Deitel et. al
 - <https://moodle.hku.hk/mod/resource/view.php?id=2266488>

What is an Operating System?

- A PROGRAM that
 - controls the execution of application programs
 - which program runs first? for how long? which program to be suspended or swapped out?
- Two main functions
 - primarily as a **resource manager**
 - Managing resources – CPUs, memory, disks, files, . . .
 - Decides between conflicting requests for **efficient** and **fair use** of resources
 - An **interface** between applications and hardware
 - **Separates** applications from the hardware they access
 - OS provides the APIs (**Application Program Interfaces**) for programs to ask for OS services/resources
 - This greatly simplifies application development

Core Services

- Allow applications to run on the system
- Allow running applications to use memory as well as share the memory
- Allow running applications to interact with each other
- Allow applications to access and share data that stored in persistent storage

The crux of our problem

- How does the operating system support these services?
 - “Note that why the OS does this is not the main question, as the answer should be obvious: it makes the system easier to use. ” “Thus, **we focus on the how**: what **mechanisms and policies** are implemented by the OS to support its services? How does the OS do so **efficiently**? What **hardware support** is needed?”

Process & Processor Management

- A process is basically a **program in execution**
 - Process needs resources to accomplish its task
 - CPU, memory, I/O, files, initial data
 - Process termination requires reclamation of any reusable resources
- Typically system has **many processes**, some are user processes, some are OS/kernel/system processes, **running concurrently** on **one or more CPUs** (cores)

Process & Processor Management

- Responsibilities of OS
 - How to manage and control application processes?
 - There are many processes running in the system
 - How to perform restricted operations?
 - Processes must be allowed to perform I/O and other restricted operations
 - How to provide the illusion of many CPUs?
 - Each given to a process
 - How to regain control of the CPU?
 - Even if processes are not being cooperative
 - How to design an effective and efficient processor scheduler?
 - Even without perfect knowledge of processes' characteristics

Memory Management

- Von Neumann model of computing
 - Instructions & data of a running process **must be** in physical memory in order to execute
- Main objective
 - With limited memory and many processes are running concurrently, OS determines what should be in memory
 - To optimize **CPU utilization** and computer **response** to users

Memory Management

- Responsibilities of OS
 - How can running processes share the single pool of physical memory?
 - Giving an abstraction of a private memory space for each process
 - How to maintain control over which memory locations an application can access?
 - We don't want other processes to write to another process's memory
 - How to run many processes with total memory demand higher than physical limit?
 - Getting help from the larger, slower hard disks
 - How to manage free space? What should OS do if running out of free space?
 - If running out of free space, getting back some memory from running processes

Concurrency

- OS is a concurrent program
 - Many internal kernel data structures may be updated concurrently by multiple execution logics
- Nowadays, multi-core systems and multi-thread programs are prevalent
 - Threads can update shared data simultaneously
- When there are many concurrently tasks running within the same memory space, how can we build a correctly working program?
 - Programs or OS have to carefully access shared data, with the uses of proper synchronization primitives, in order to work correctly

Concurrency

- Responsibilities of OS
 - What kind of primitives are needed for synchronization?
 - We may have to examine different kinds of synchronization and concurrency issues
 - What support do we need from **hardware** and **OS** in order to build useful synchronization primitives?
 - We want the primitives to work correctly and efficiently
- How can we use them to solve concurrency problems?

File Management

- The component in OS that manages storage disks is called the file system
 - Responsible for storing any data the user created in a **reliable and efficient** manner on the disks of the system
- File System
 - Provides a uniform, logical view of information storage
 - Abstracts physical properties to logical storage unit - **file**
 - **Maps** files onto physical media and **provides mechanisms** for applications to manage and access files

File Management

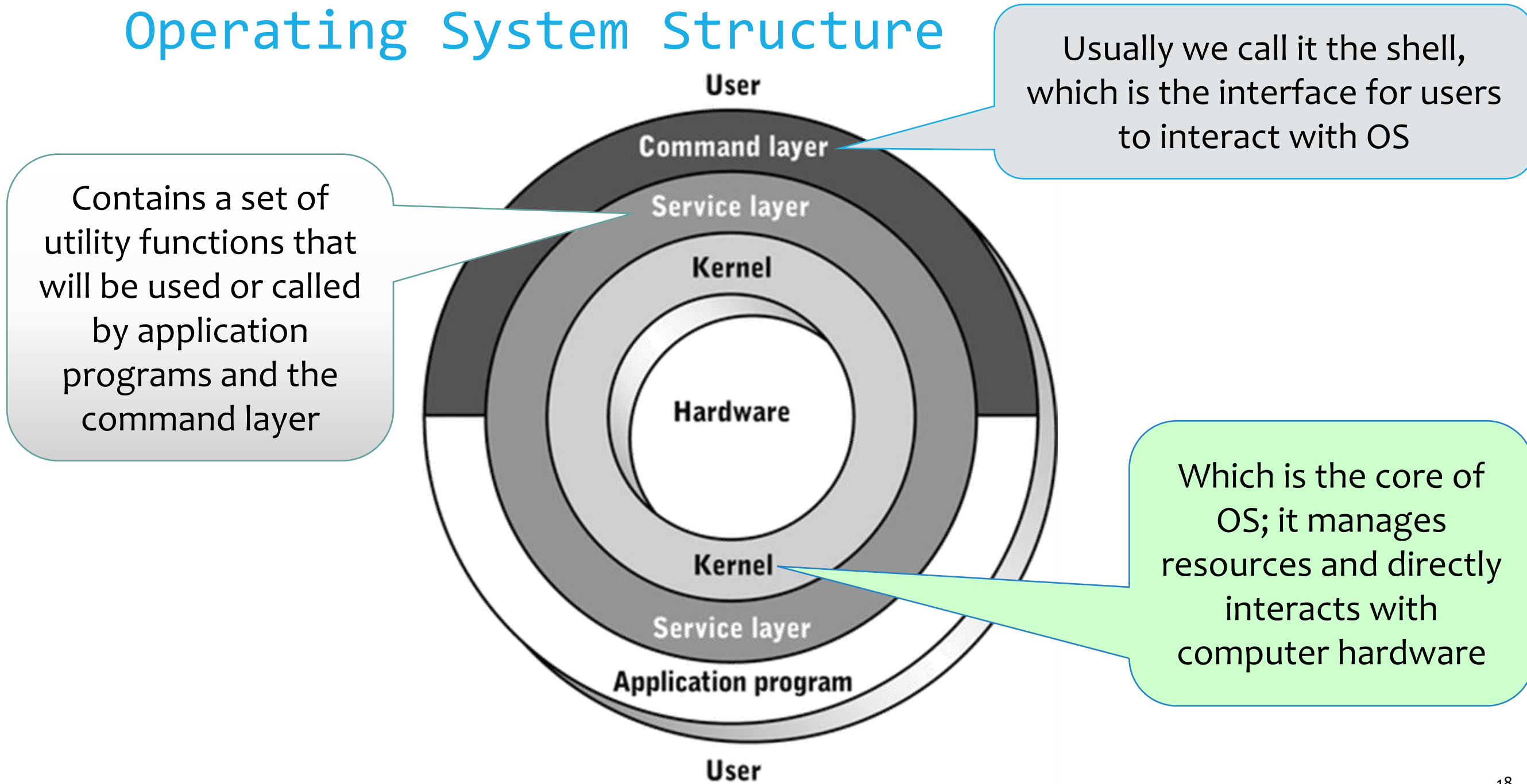
- Responsibilities of OS
 - How to manage a persistent device?
 - For example, where to find suitable storage units for a newly created file? what will happen if deleting a file?
 - How to implement the file system?
 - We need some data structures on the disk for the file system to work correctly and efficiently; e.g., given a file name, how to access the data of the file?
 - How to reduce file system I/O costs?
 - Disk access is slow, is there any way to make file access faster??

Goals of OS

- Makes the system convenient and easy to use
- Provides protection between applications and between OS and applications
- Allows computer system resources to be used in an efficient manner
- Can operate on many hardware configurations
- Provides a high degree of reliability and will not fail due to isolated application/hardware errors
- Protect resources from unauthorized access by users and software

OS Architectures

Operating System Structure



Architectures

- Operating Systems tend to be complex
 - Provide many services and support variety of hardware and software
- Operating system architectures help manage this complexity
 - Organize operating system components (functionalities)
 - **Specify the privilege level** with which each component executes
- Common options
 - Monolithic Architecture
 - Layered Architecture
 - Microkernel Architecture
 - Modular Approach

User Mode / Kernel Mode

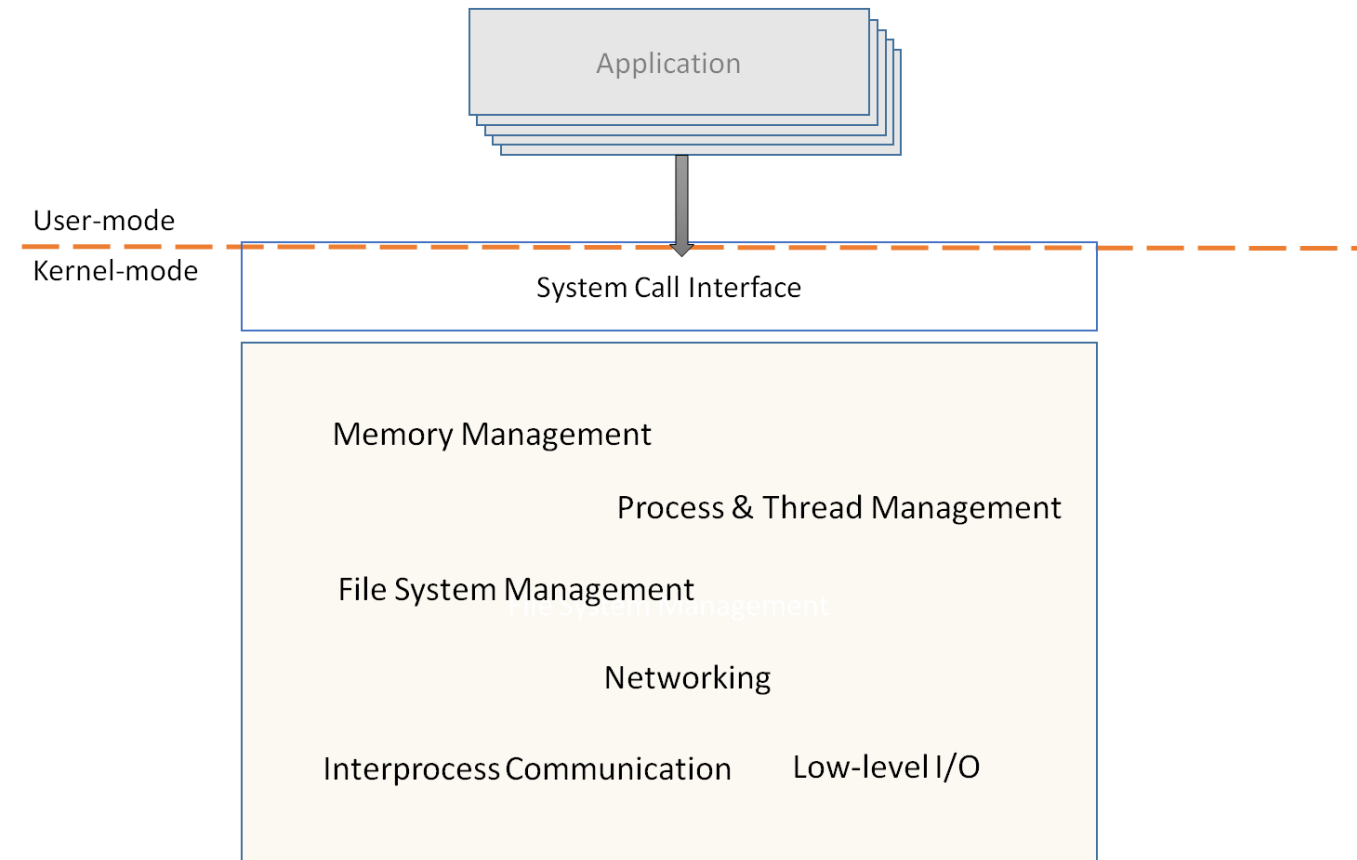
- CPU operates in (**at least**) two modes
 - Kernel mode
 - Also known as privilege mode or supervisor mode
 - CPU is running in kernel mode when the OS kernel executes
 - It is at the "highest" privilege level and can execute any instruction the machine is capable of executing
 - User mode
 - CPU is put in user mode when a user application is executing
 - Running in a lower privilege level
 - Only a subset of the machine instructions is available
 - Ensures that one user program cannot execute instruction that may interfere with operation of other user programs

System Calls

- A system call is mechanism allows an application program to make a request to the OS **for resources/services**
 - the set of system calls is the **interface (API) to the services** provided by the OS
- When a system call occurs, the system **switches** from **user mode** to **kernel mode** and executes the corresponding **kernel's system function**
 - We call this **mode switch**
 - This is achieved by some special instructions that helps the process to **trap** into the kernel and **return-from-trap** back to user mode

Monolithic Architecture

- Every OS component is contained **in the kernel**
 - Traditionally, OS code **did not** structure its components as modules with clearly defined interfaces
 - All its components are interwoven into **one large program** that **runs in privilege mode**
 - Any component (within kernel) can **directly communicate** with any other (e.g. by using function calls)
 - Data structures are **easily shared** as all in one program

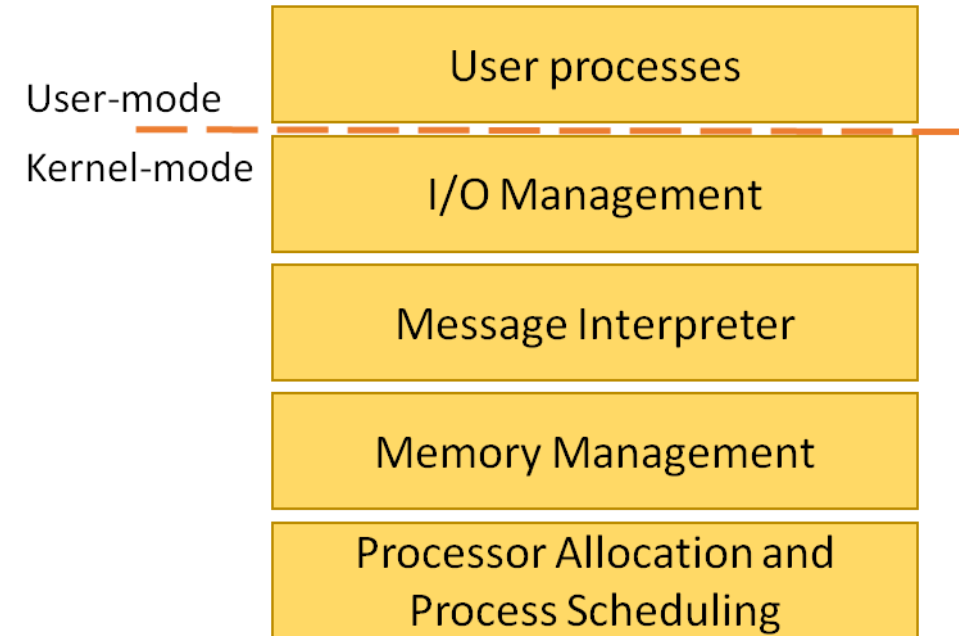


Monolithic Architecture

- Adv: Tend to be **highly efficient**
- Disadvantages
 - Components can interact with hardware directly
 - Architecture dependent code was spread throughout the kernel
 - Components can access each other's data and functions directly
 - Changes made in one component could affect other components
 - Bugs in one component can adversely affect another component
 - Difficulty in determining source of subtle errors
 - More susceptible to damage from malicious code (component)

Layered Architecture

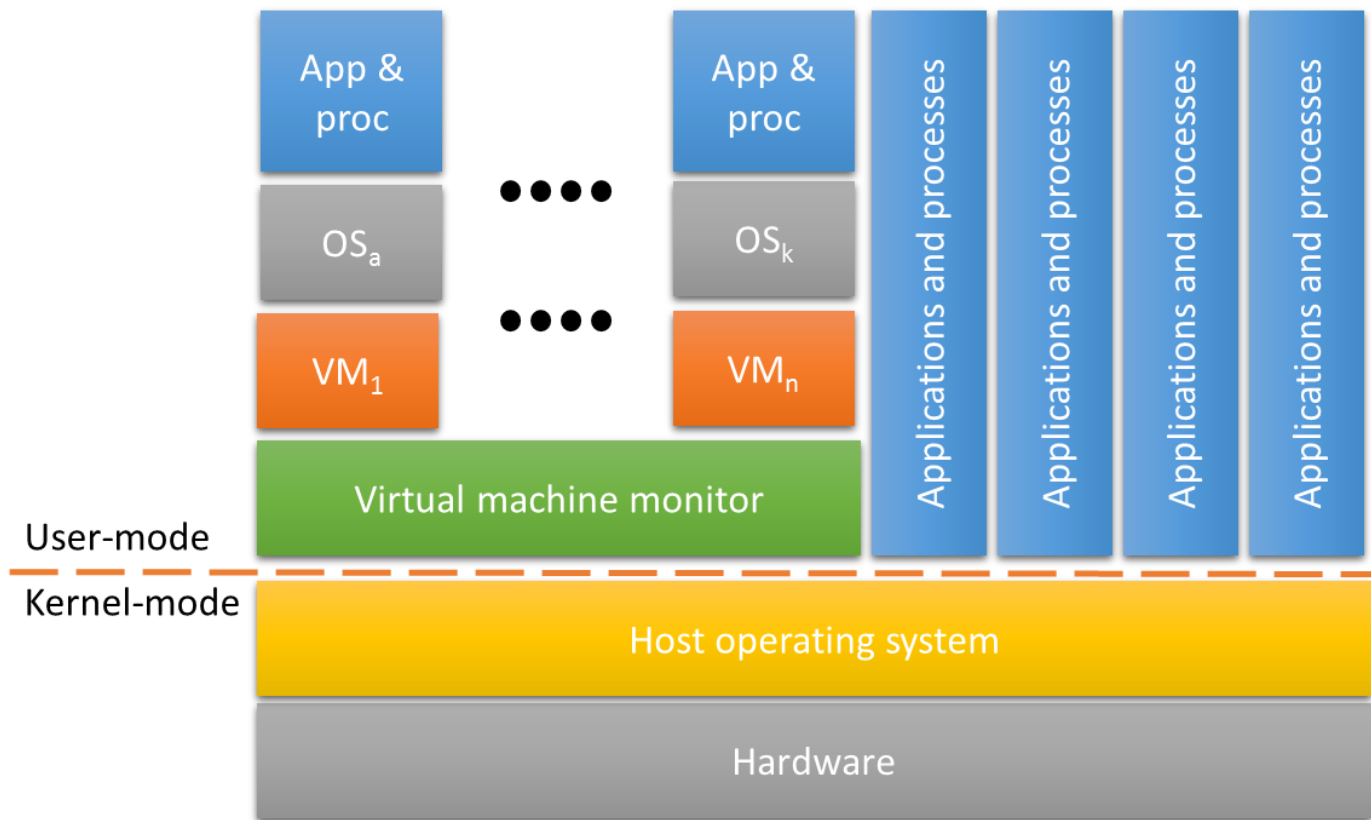
- Groups components that perform similar function or specific role into a module and layers the modules one on top of the other
- Each layer **communicates only** with layers immediately above and below it
 - Processes' requests might pass through many layers before being serviced
 - **System efficiency can be less** than monolithic kernels



Layered Architecture

- Pros
 - Simplicity of construction and debugging
 - Allows debugging starting at lowest layer, adding one layer at a time until the whole system works correctly
 - **Information hiding** – each layer only knows the interface provided by immediate lower layer
 - We can replace one layer by another implementation without affecting other parts of the system
- With the layered approach, the designers have a choice where to draw the kernel-user boundary

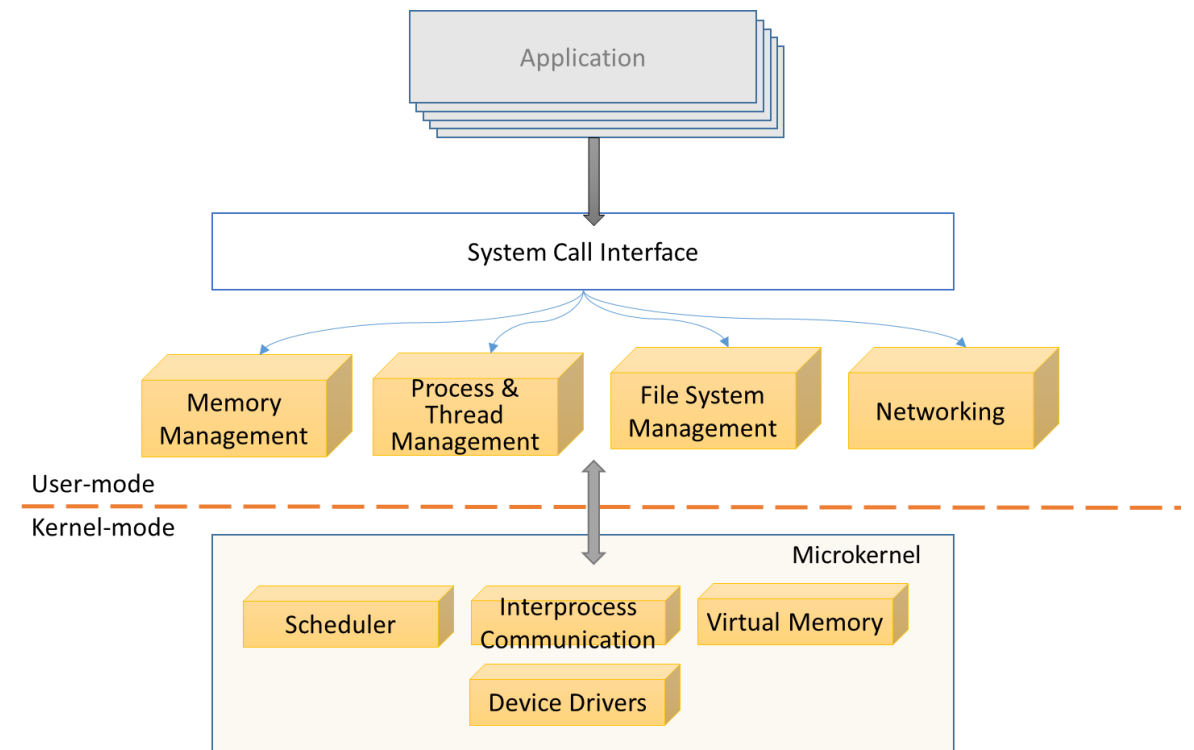
Virtual Machine – An Example of Layered Approach



- Virtual machine monitor
 - Also named as *hypervisor*
 - Runs on top of host OS – in user mode OR incorporated into host OS – kernel mode
 - Virtualizing the hardware resources and giving the illusion to each running OS that it controls the machine (which is a virtual machine)
 - In essence, it serves as an OS for OSs.

Microkernel Architecture

- **Moves** as much functionalities from the kernel space into user space processes (being called as servers)
 - Attempt to keep kernel small
 - The system becomes more stable as only the **bare essentials** are running in kernel mode
 - More easy to extend and port to other platforms

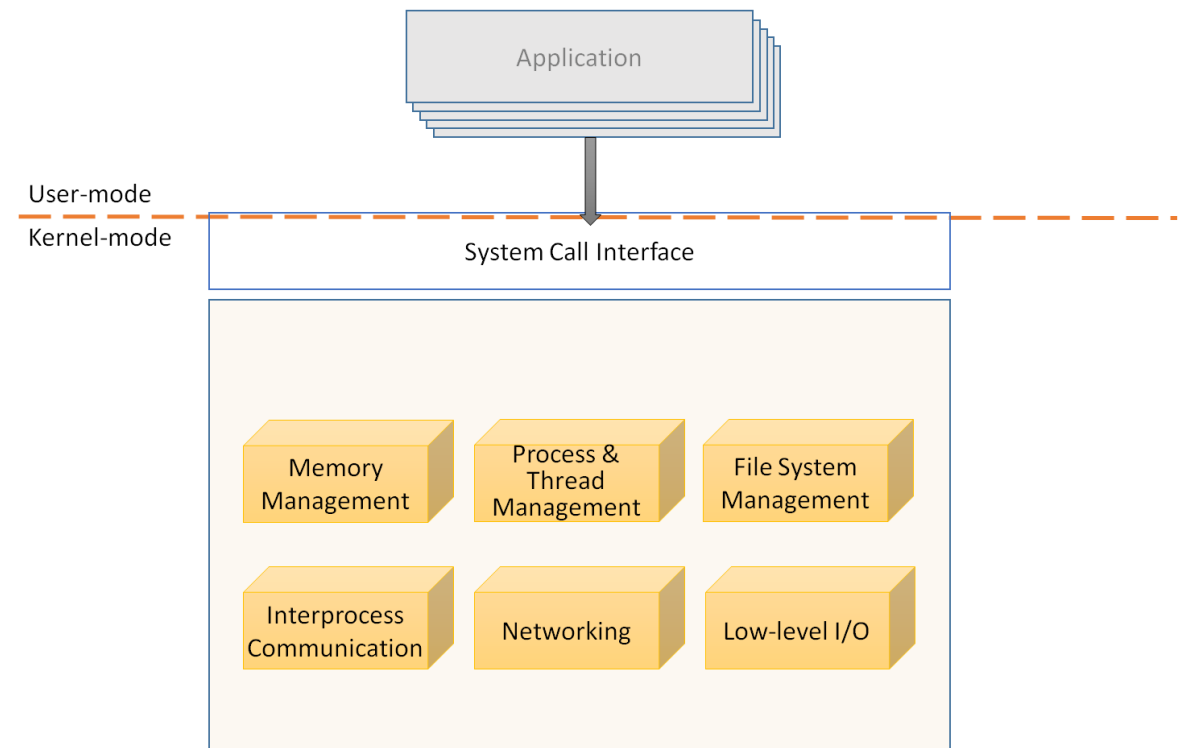


Microkernel Architecture

- Pros
 - Extensible, portable and scalable
 - More secure & reliable (less code is running in kernel mode)
- Con
 - Server processes (in user space) are interacting by means of message exchanges, which need the helps from kernel
 - **Induce a significant performance overhead** because of communications have to go through kernel

Modular Approach

- Most modern operating systems still adopt the monolithic architecture but implement kernel modules
 - Each core component is separated and implemented as module
 - The whole kernel is a collection of modules

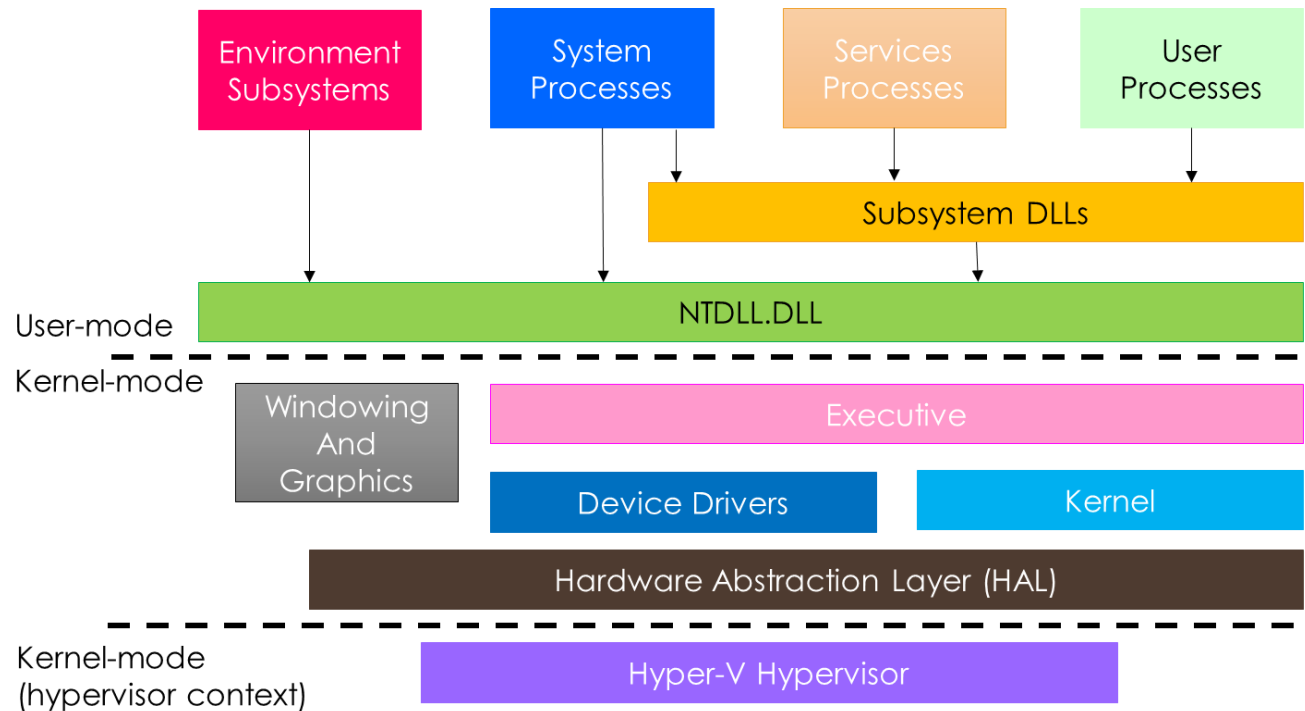


Modular Approach

- Overall, similar to layers but is more flexible and efficient as
 - each kernel module has well-defined, **protected interfaces**; any module can call any other module
 - more efficient to communicate between modules as they are all in kernel
 - allows certain features to be implemented dynamically and loaded as needed (dynamically loadable modules)
 - Memory is conserved as only required modules are loaded in memory
 - more extensible as kernel modules can be modified separately and new modules can be added easily
- Examples: Solaris, Linux, and traditional Unix kernels

Examples

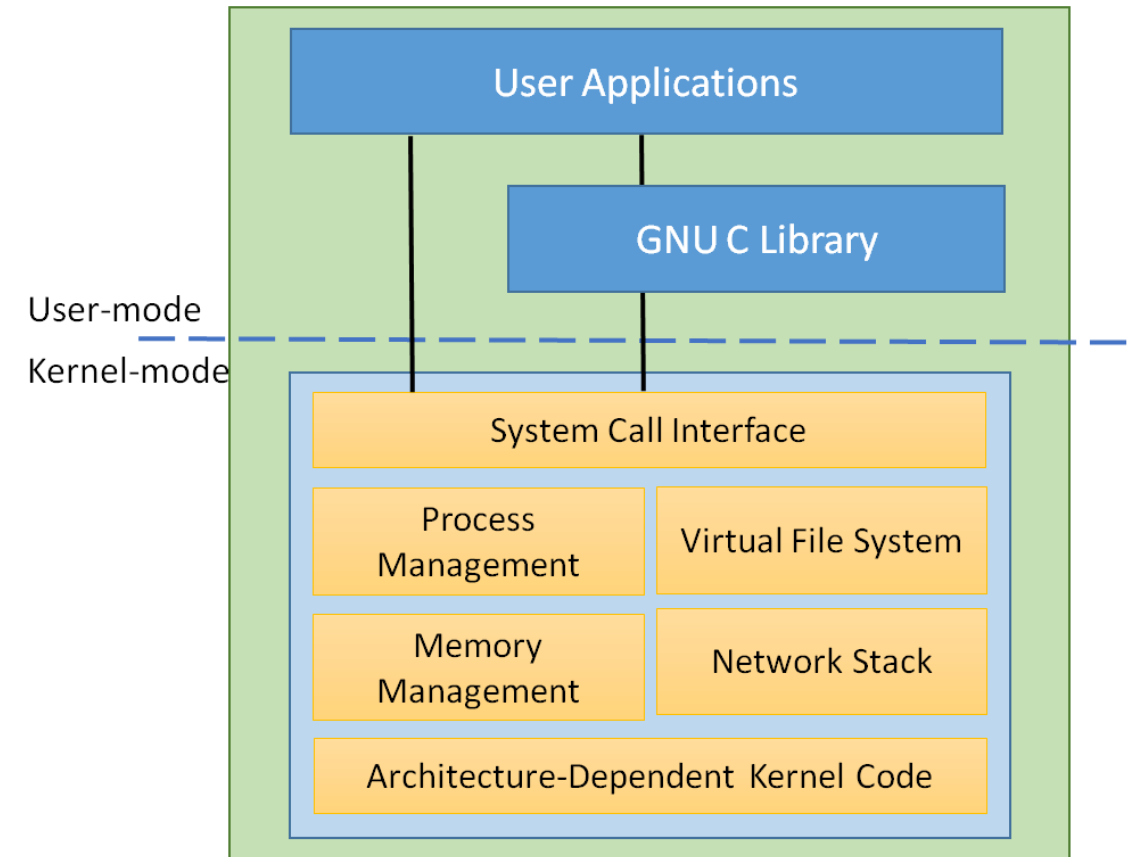
Windows System Architecture



(Source: Windows Internals Part 1)

A hybrid kernel with modular design principles

Linux System Architecture

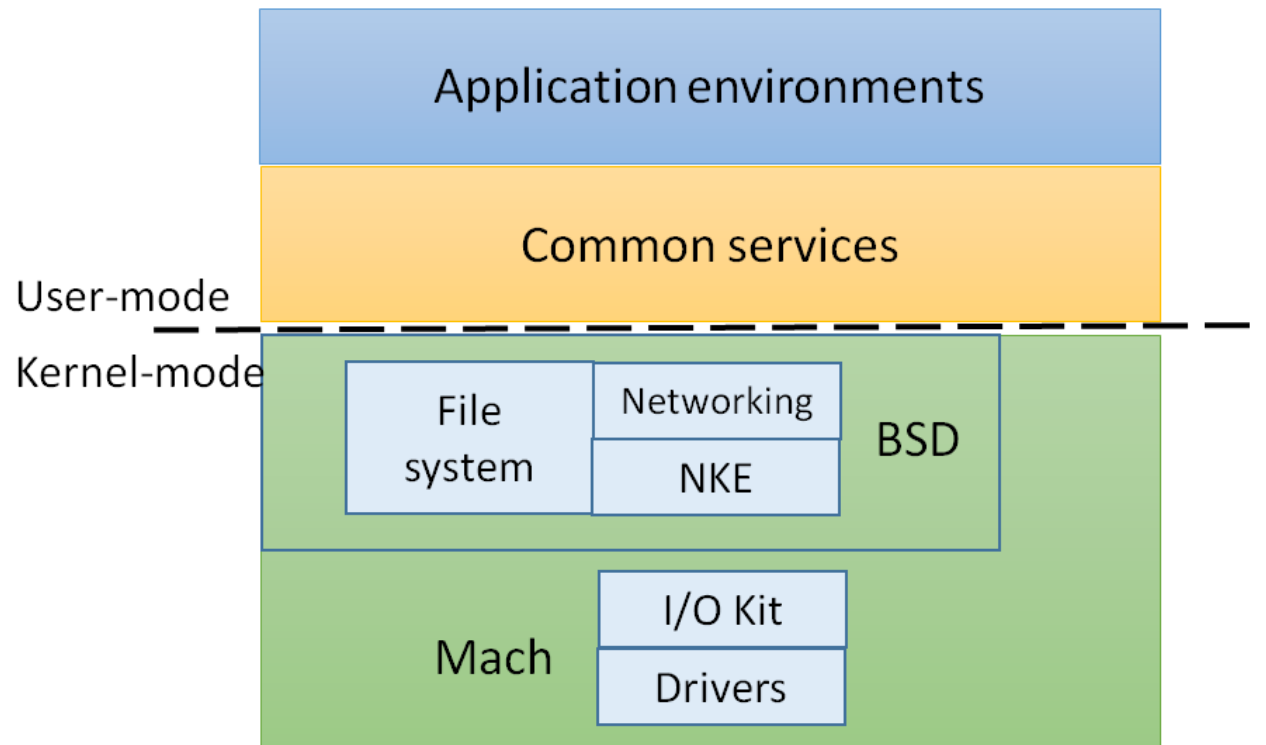


(Source: Anatomy of the Linux Kernel by M. Tim Jones)

A monolithic approach with the support of loadable kernel modules

Examples

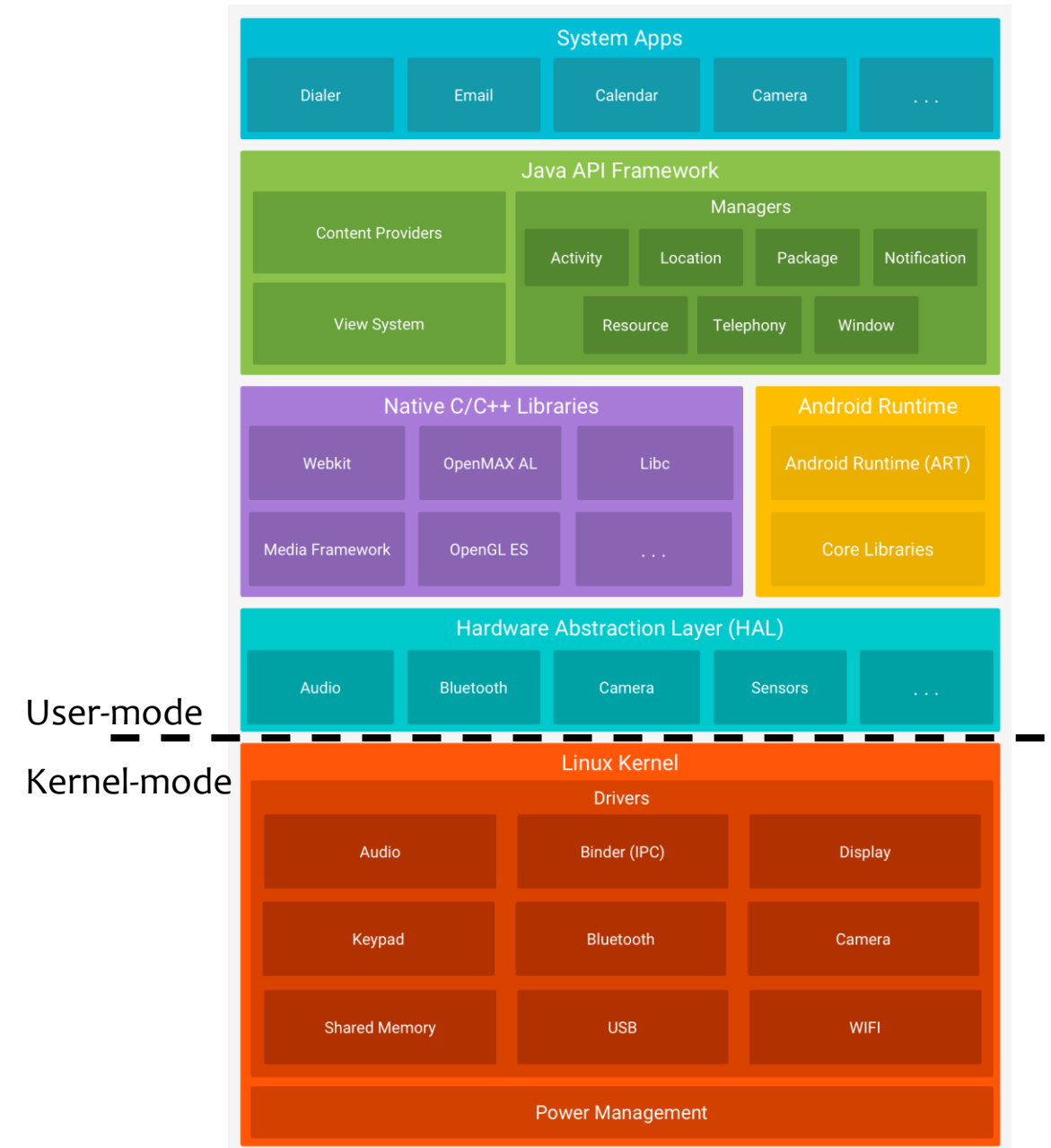
Mac OS X Architecture



(Source: Kernel Programming Guide by Apple)

A hybrid kernel with the support of loadable kernel extensions

Android System Architecture



(Source: Android Developers)

Summary

- What is OS? It is a resource manager that manage & coordinate the use of resources amongst users, and it also provides an abstractive view on the computer to users.
- Different OS architectures have their advantages and disadvantages; however, modern OSs tend to use the monolithic approach with layers and kernel modules for extensibility.