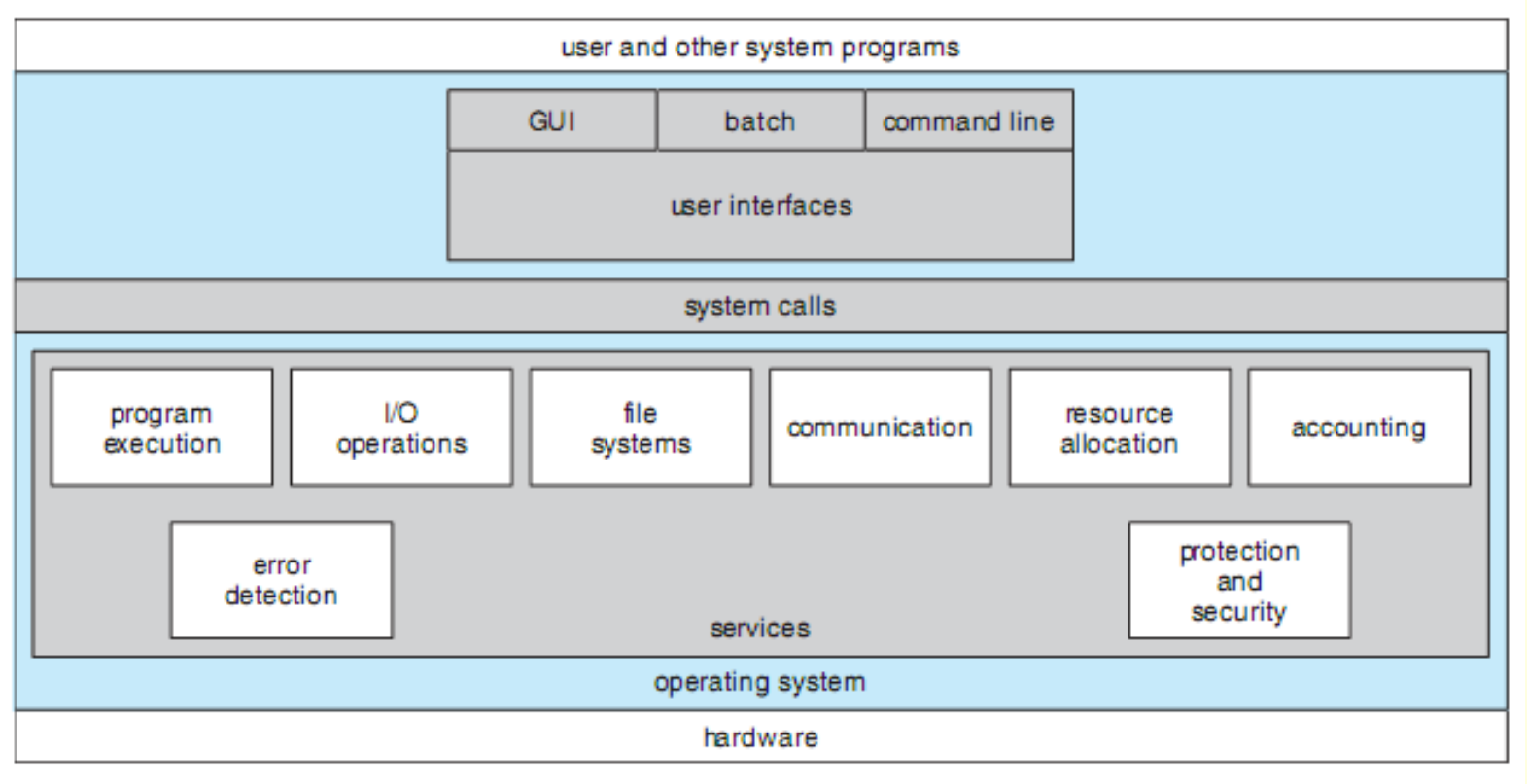
Operating System Structures

Mehdi Kargahi
School of ECE
University of Tehran
Fall 2016

Outline

- What services are provided by the OS for
  - Users
  - Processes
  - Other systems
- Different operating system structures
- Operating systems: installing and booting

OS Services

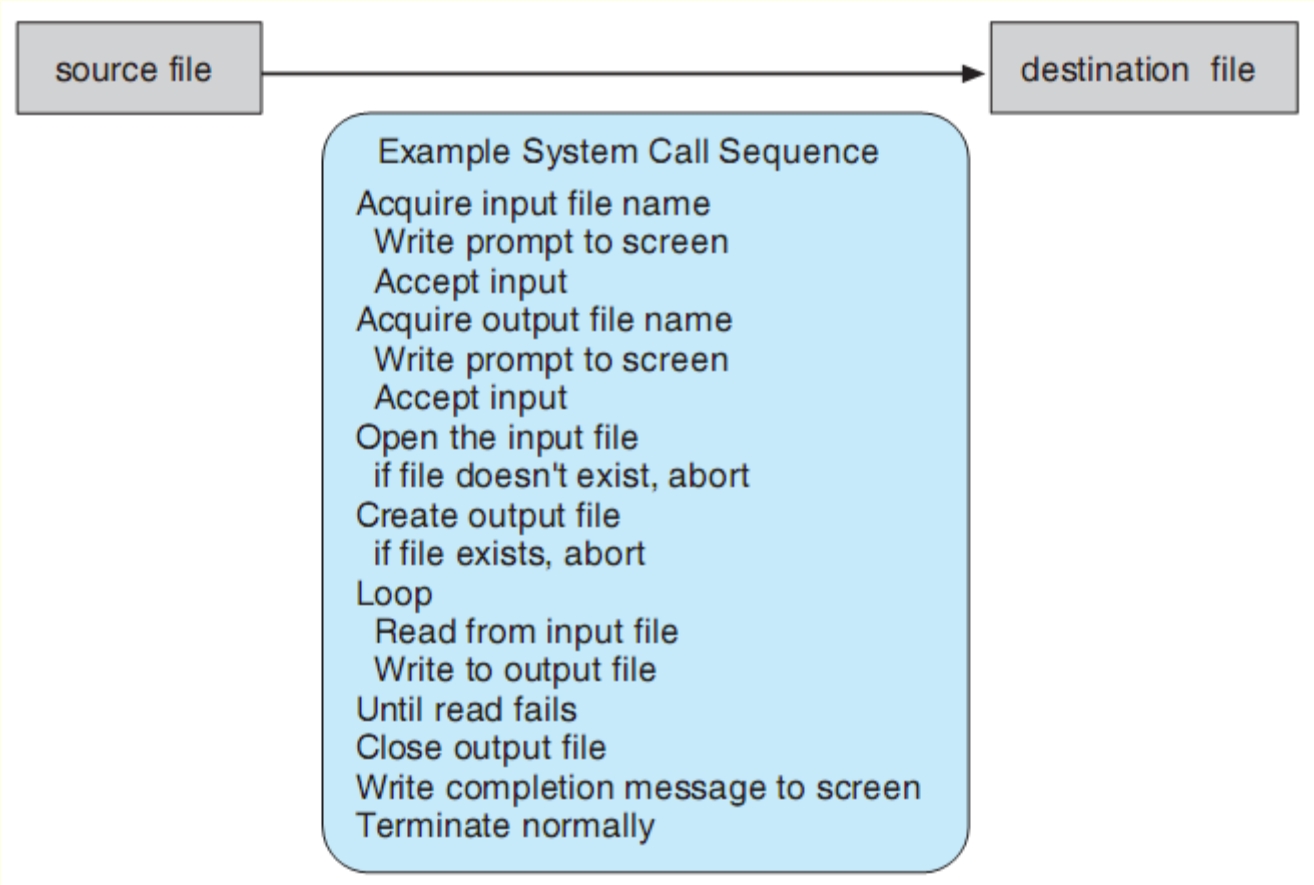

user and other system programs
GUI
batch
command line
user interfaces
system calls
program execution
I/O operations
file systems
communication
resource allocation
accounting
error detection
protection and security
services
operating system
hardware

User OS Interface

- User interface (UI)
  - Command-Line Interface (CLI)
    - In the kernel
    - As a special program (e.g., a shell)
  - Batch interface (file)
  - Graphical User Interface (GUI)

System Calls

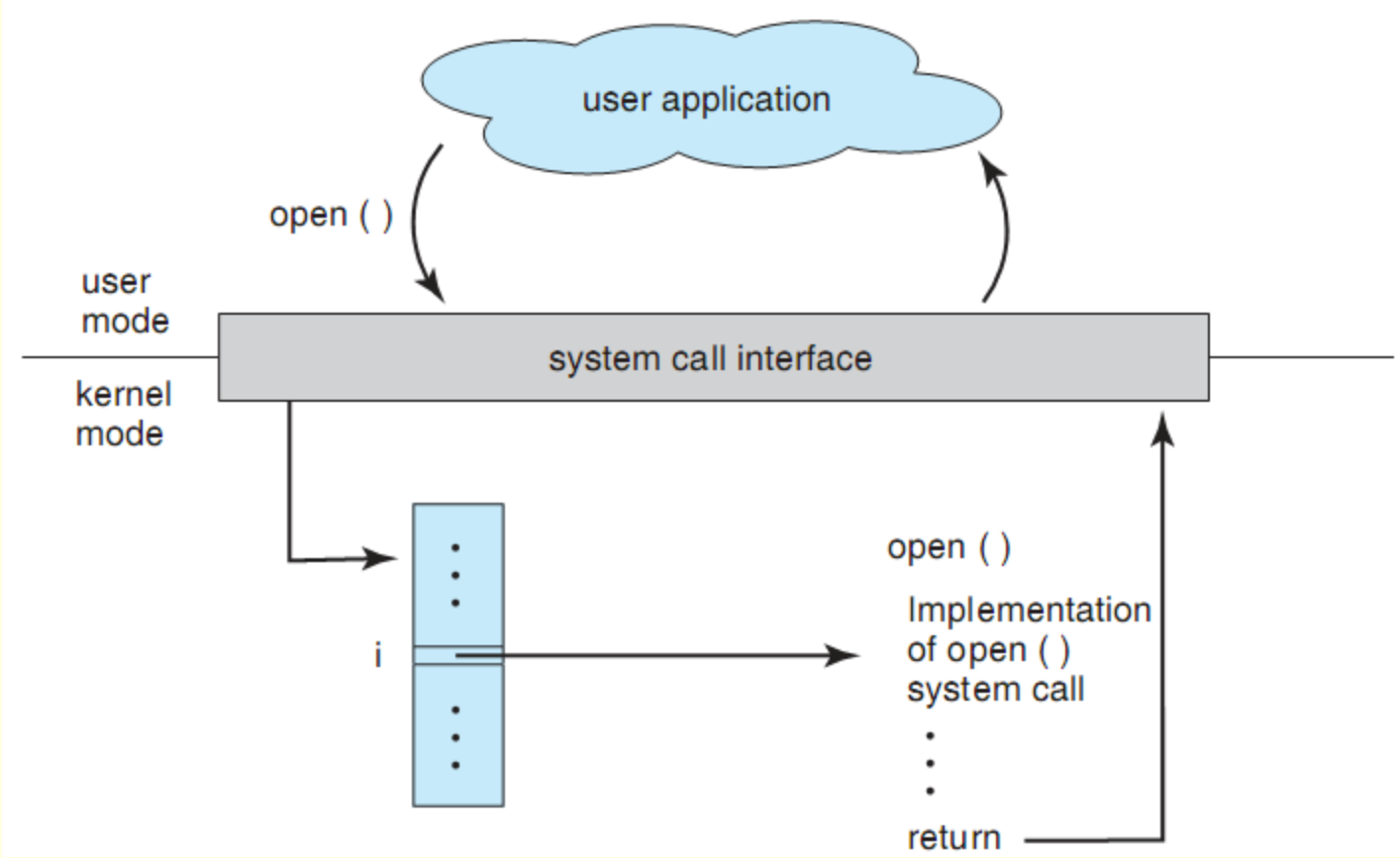
- The basic way to use the OS services
- Each OS has its own system calls



Application Programming Interface (API)

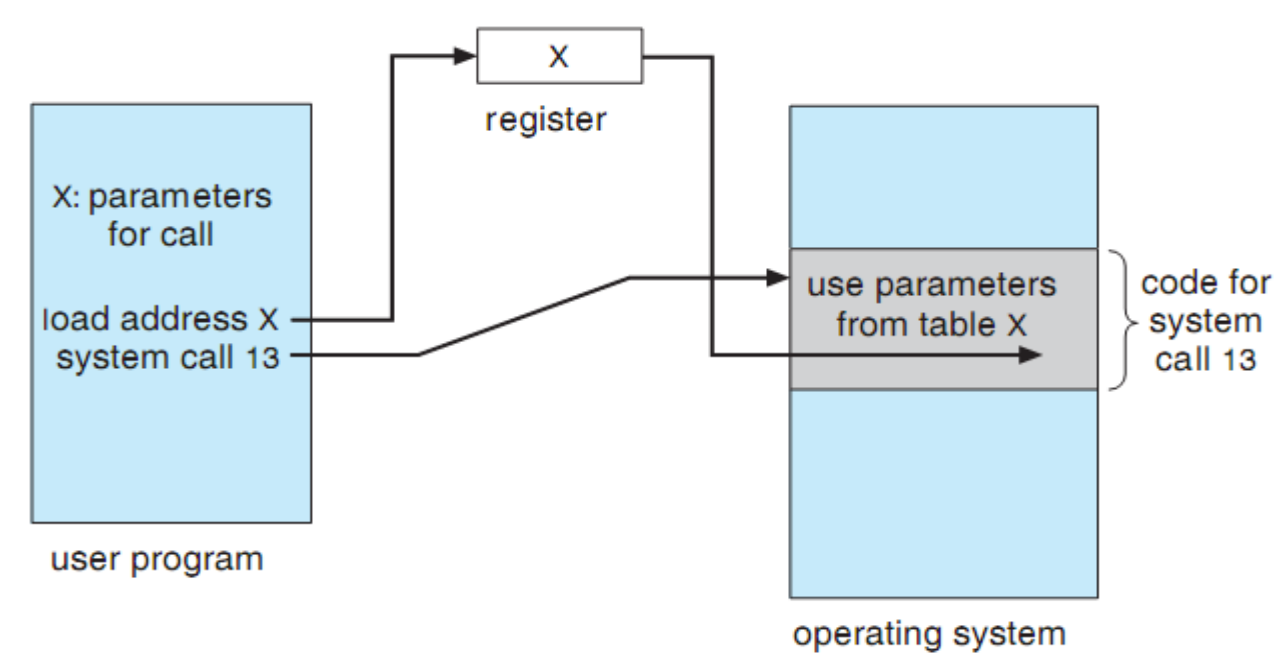
- Standardizing the use of system calls
- Portability of programs on every system supporting the same API (*sending parameters and getting return values*)
  - Win32 API (Windows)
  - POSIX API (UNIX, LINUX, MAC OS X)
  - JAVA API (for programs running on JVM)
- System programs: sequences of system calls to perform more complex operations
  - File management, program loading, …

Handling a System Call



Methods of Passing Parameters

- Registers
- Blocks or tables in memory



- Stack

Types of System Calls

EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS

	Windows	Unix
Process Control	CreateProcess()	fork()
	ExitProcess()	exit()
	WaitForSingleObject()	wait()
File Manipulation	CreateFile()	open()
	ReadFile()	read()
	WriteFile()	write()
	CloseHandle()	close()
Device Manipulation	SetConsoleMode()	ioctl()
	ReadConsole()	read()
	WriteConsole()	write()
Information Maintenance	GetCurrentProcessID()	getpid()
	SetTimer()	alarm()
	Sleep()	sleep()
Communication	CreatePipe()	pipe()
	CreateFileMapping()	shm_open()
	MapViewOfFile()	mmap()
Protection	SetFileSecurity()	chmod()
	InitlializeSecurityDescriptor()	umask()
	SetSecurityDescriptorGroup()	chown()

Types of System Calls

- Communications
  - Message-passing model: (host name, process name)→ source: client, receiving daemon: server
    - Suitable for transferring smaller amount of data
    - Simpler implementation of communication among computers
  - Shared-memory model:
    - maximum transfer rate
    - Problems: synchronization and protection

System Programs

- System programs, also known as system utilities, provide a convenient environment for program development and execution.
- They can be divided into these categories:
  - File management
  - Status information
  - File modification (e.g., text editors)
  - Programming-language support (e.g., compilers)
  - Program loading and execution
  - Communications
  - Background services (e.g., process scheduler)

Operating System Design and Implementation

- Design Goals
  - At the highest level: batch, time-shared, single-user, multi-user, distributed, real-time, …
    - User goals: simple and easy to learn and use, reliable, safe, fast
    - System goals: easy to design, implement, and maintain, flexible, reliable, error free, efficient
  - NO UNIQUE solution for selecting the best requirements among the above

Some definitions

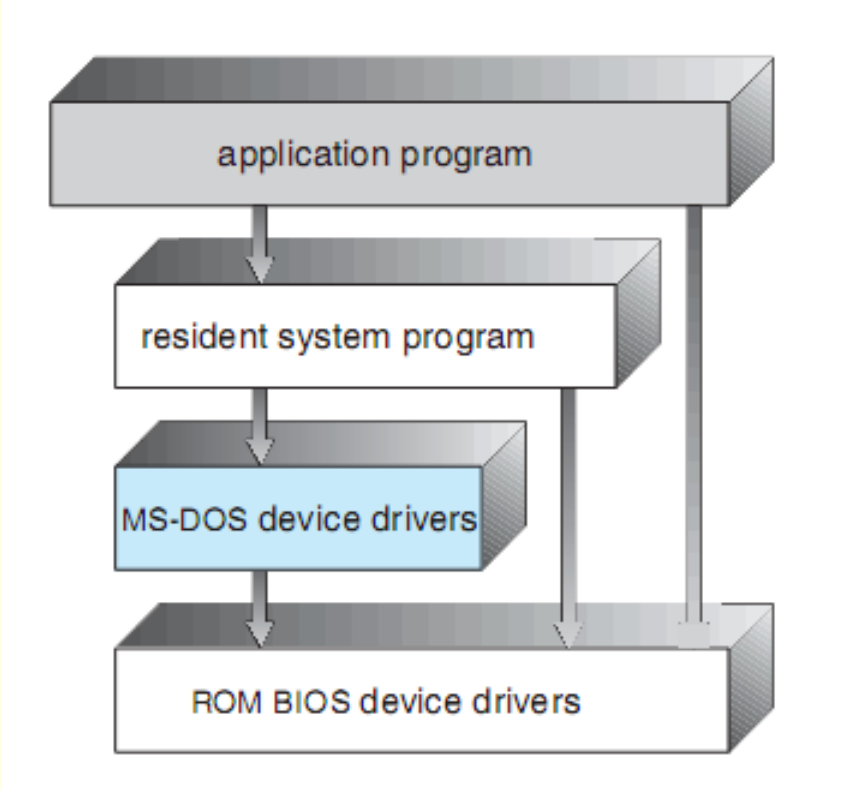
- Mechanism: How to do something
- Policy: What will be done
- Implementation of operating systems
  - Traditionally, written in assembly language
  - Currently, mostly written in C or C++

Operating System Structure

- Simple structure
- Layered approach
- Microkernels (μ-kernels)
- Modules
- Virtual Machines

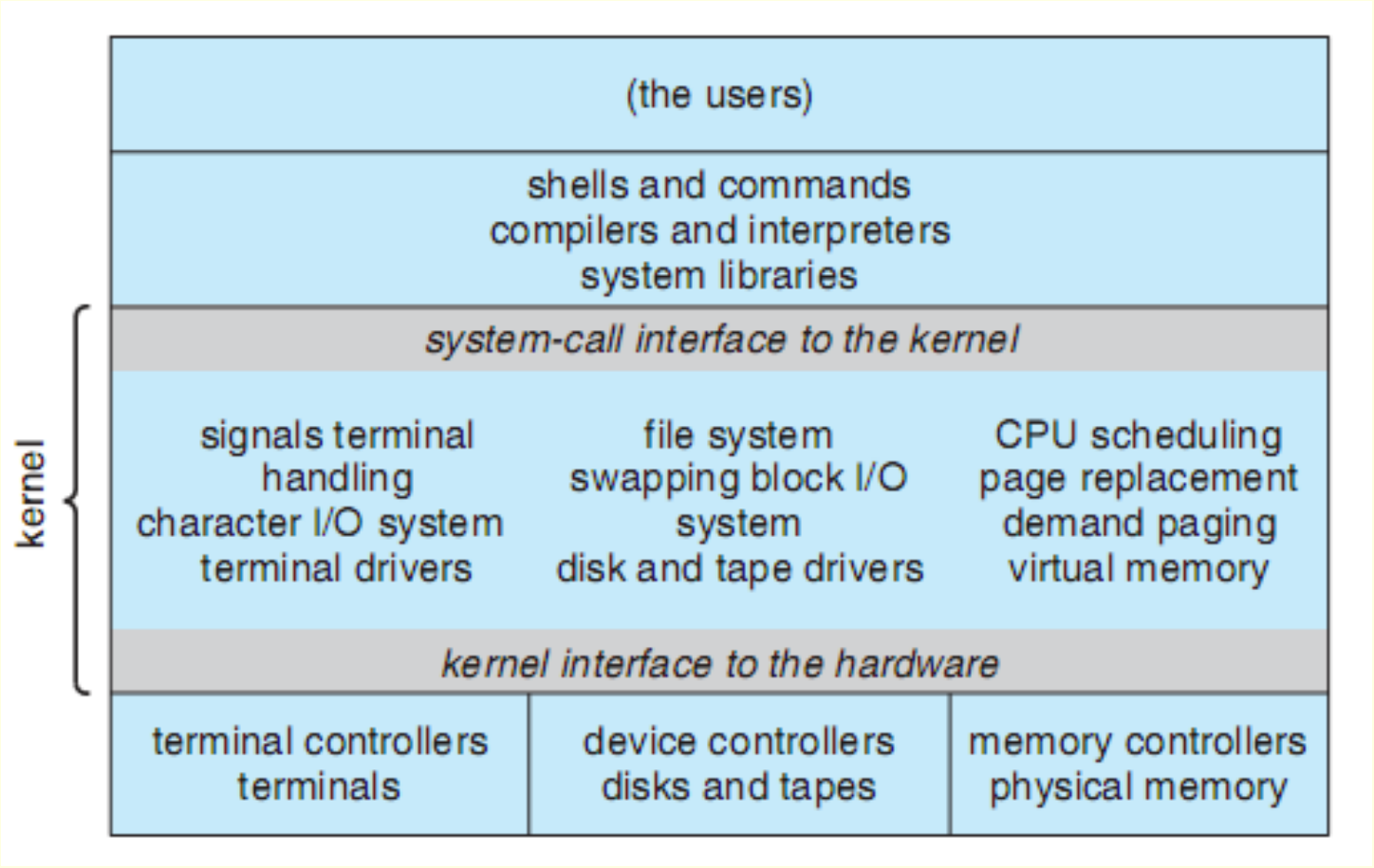
Simple Structure

- MS-DOS layer structure

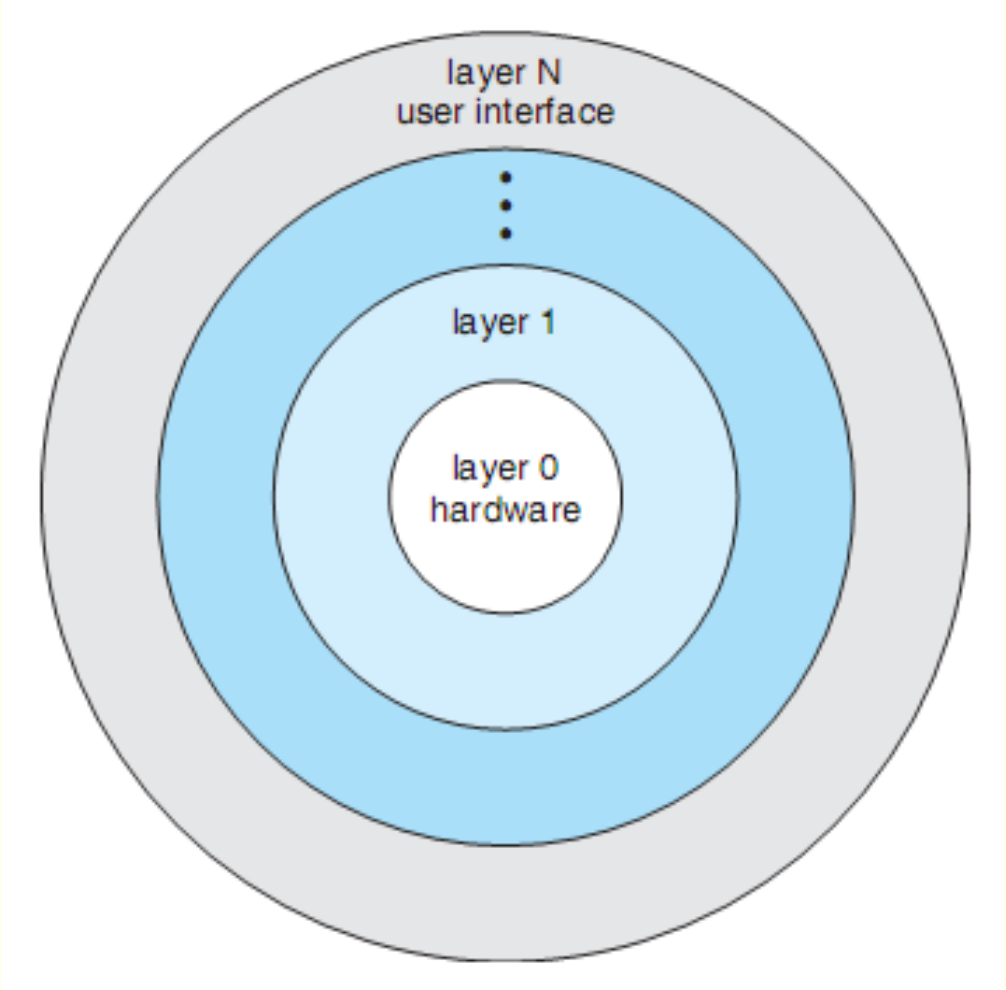


Simple Structure

- UNIX partially-layered structure



Layered Approach

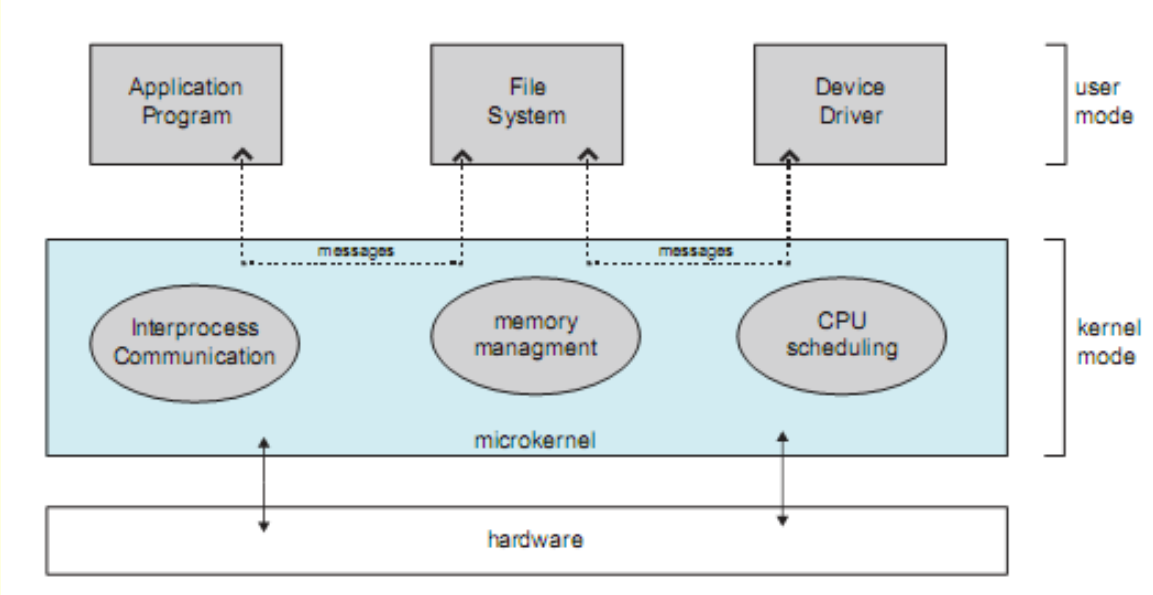


Layered Approach

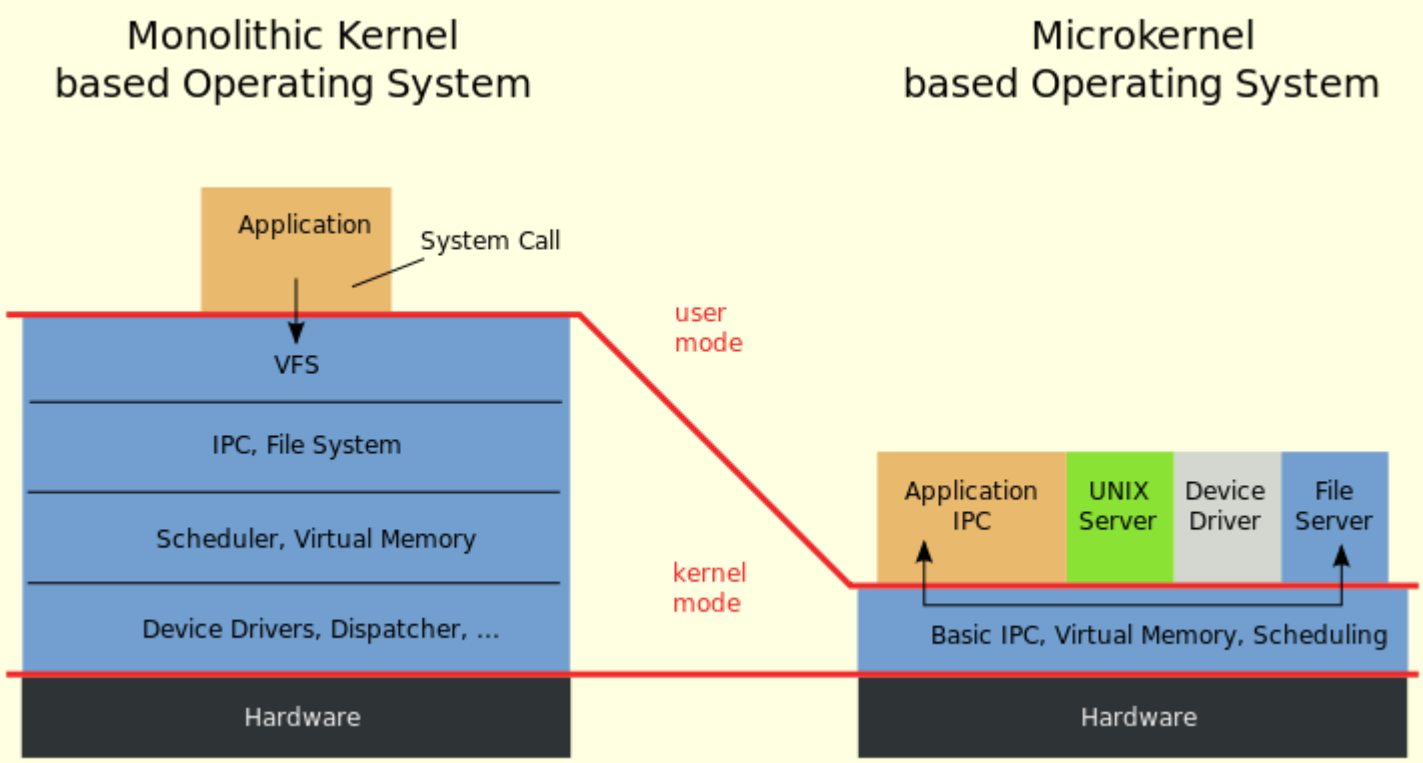
- Properties
  - Simplicity of construction
  - Simplicity of Debugging
- Problems
  - Precise definition of layers
    - Example: Memory manager requires device driver of backing store (due to virtual memory)
    - The device driver requires CPU scheduler (since if the driver waits for IO, another task should be scheduled)
    - CPU scheduler may require virtual memory for large amount of information of some processes
  - Less efficiency: due to the number of layers a request should pass

Microkernels

- Removing all nonessential components from the kernel
- Implementing them as system and user-level programs
  - C/S model & the way of finding services
- What μ-kernel does?
  - Minimal process and memory management
  - Communication facilities
    - IPC: Inter-Process Communication



Monolithic Kernal vs. Microkernel



Microkernels

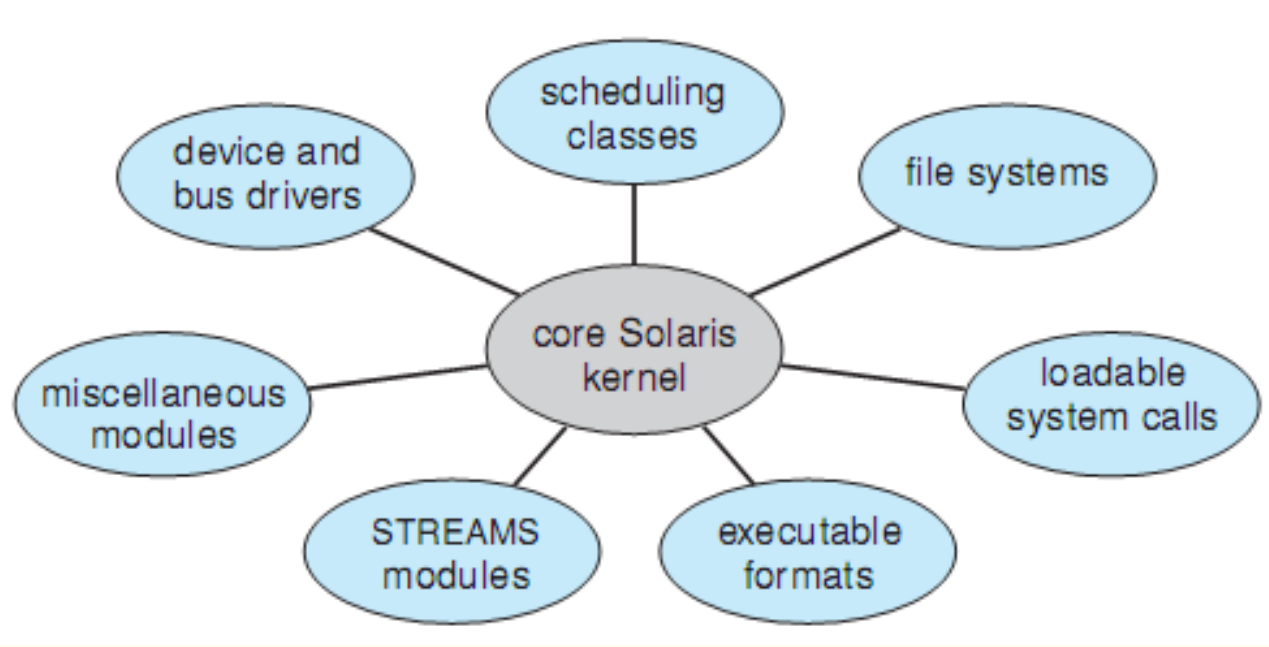
- Advantages
  - Extensibility of the OS
  - Portability
  - Potential for making distributed services
  - More security and reliability (service failure doesn't destroy OS)
- Disadvantages
  - Performance loss (due to IPC)
- Mach, QNX, L4, the first release of Windows NT, …

Modules

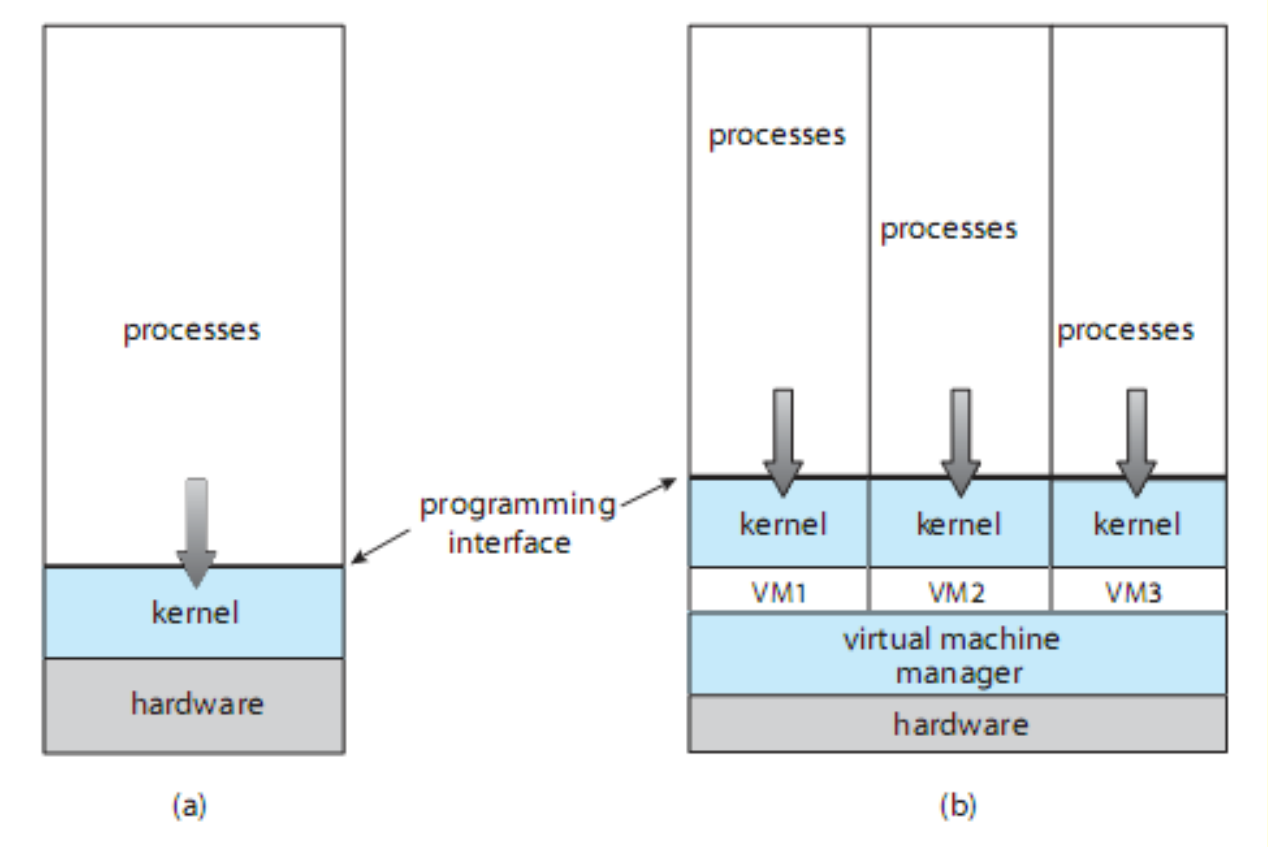
- One of the best current methodologies
  - Using OOP techniques to create a modular kernel
- Kernel: a set of core components
- Dynamically links for additional services (dynamically loadable modules)
  - Boot time or run-time
- Exploits the benefits of both layered and μ-kernel structures:
  - Each module has defined protected interfaces
  - Primary module has only core functions
  - Modules do not need to use message passing to communicate

Modules

- Solaris loadable modules



Virtual Machines



Virtual Machines

- Benefits
  - VM are completely isolated➔ no protection problems
  - No direct sharing of resources
  - Sharing the same HW among different environments with different operating systems running concurrently
  - It is a perfect vehicle for OS research and development
- Main difficulty
  - Disk space➔ solution: minidisks

Virtual Machines

- Implementation
  - User mode
    - Virtual user mode
    - Virtual kernel mode
  - Kernel mode
- The underlying OS may be structured as a layered, μ-kernel, …
- Major difference is time!
  - Real I/O might take 100 ms
  - Virtual I/O might take
    - Less time (because it is spooled)
    - More time (because it is interpreted)

Hybrid Systems

- Very few operating systems adopt a single, strictly defined structure
  - Instead, they combine different structures, resulting in hybrid systems that address performance, security, and usability issues
- Linux and Solaris:
  - Monolithic, because having the operating system in a single address space provides very efficient performance.
  - However, they are also modular, so that new functionality can be dynamically added to the kernel
- Windows:
  - Largely monolithic
  - It retains some behavior typical of microkernel systems,
  - Also provide support for dynamically loadable kernel modules

OS Debugging

- Debugging: Activity of finding and fixing errors in a system, both in hardware and in software.
- Performance problems are considered bugs, so debugging can also include performance tuning, which seeks to improve performance by removing processing bottlenecks
- Core dump: a capture of the memory of the process
- Crash: A failure in the kernel (Crash dump)