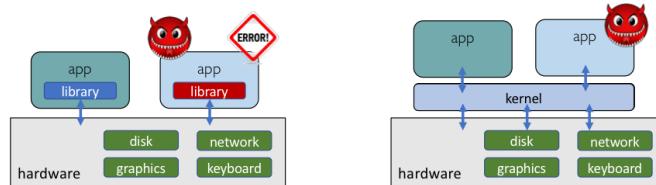


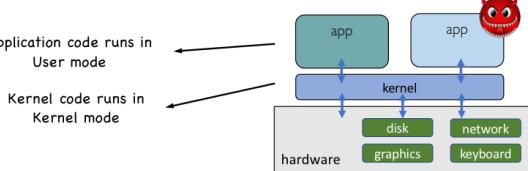
Evolution of Operating Systems

- A library to handle low-level I/O
 - Issue: Fault and security isolation
- Kernel: A bigger “library” to handle low-level I/O
 - Kernel needs to be protected from faulty/malicious apps



Kernel Mode vs. User Mode

- Dual-mode operation allows OS to protect itself and other system components
 - Mode bits provided by CPU hardware
 - Provides ability to distinguish when system is running user code or kernel code
 - Some instructions designated as privileged, only executable in kernel mode



Dual-mode Operation

- Hardware provides at least two modes:
 - “Kernel” mode: Run kernel code
 - “User” mode: Normal programs executed
- What is needed in the hardware to support “dual mode” operation?
 - A bit of state (user/system mode bit)
 - Certain operations / actions only permitted in system/kernel mode
 - In user mode they fail or trap
 - User → Kernel transition sets system mode AND saves the user PC
 - Operating system code carefully puts aside user state then performs the necessary operations
 - Kernel → User transition clears system mode AND restores appropriate user PC

Mode Bits in CPUs

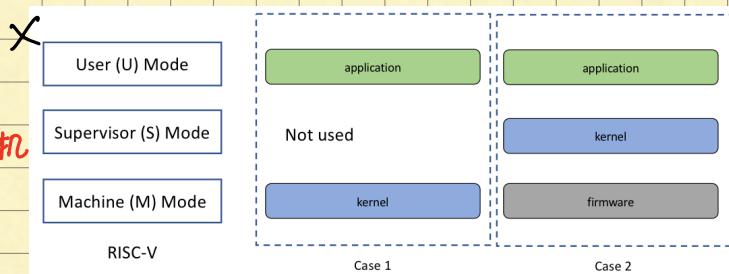


内核将 app 与 hardware 分隔，保护 hardware 不受恶意 app 危害。

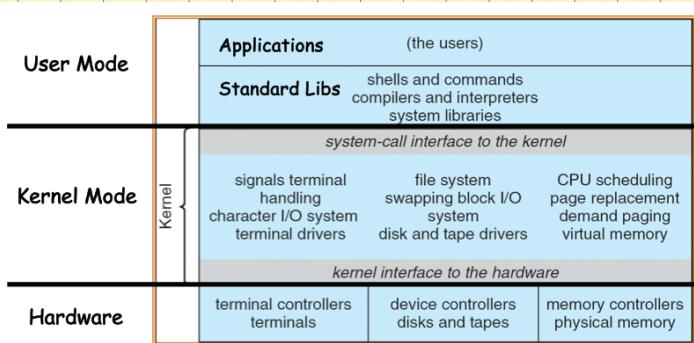
这两种模式是 OS 的
Mode bit 由 CPU 硬件提供

硬件对 dual mode 的支持：

有一个 bit 表示当前的 mode
User ↔ Kernel 模态之间的转换
保存/恢复 User PC



UNIX System Structure

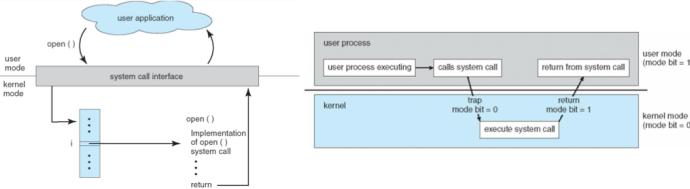


3 Types of Mode Transitions : User → Kernel

- System call
 - Process requests a system service, e.g., exit
 - Like a function call, but "outside" the process
 - Does not have the address of the system function to call
 - Marshall the syscall id and args in registers and exec syscall
- Interrupt
 - External asynchronous event triggers context switch
 - e.g., Timer, I/O device
 - Independent of user process
- Trap or Exception
 - Internal synchronous event in process triggers context switch
 - e.g., Protection violation (segmentation fault), Divide by zero, ...

System Calls

- Programming interface to the services provided by the OS
- Typically written in a high-level language (C or C++)



System Call Implementation

- Typically, a number associated with each system call
 - System-call interface maintains a table indexed according to these numbers
- The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values
- The caller needs to know nothing about how the system call is implemented
 - Just needs to obey calling convention and understand what OS will do
 - Most details of OS interface hidden from programmer by library API
 - Managed by run-time support library (set of functions built into libraries included with compiler)

Types of System Calls

- Process control
- File management
- Device management
- Information maintenance
- Communications
- Protection

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

系统调用

中断

外部异步事件导致的上下文切换 (外部设备发出通知) 与用户进程无关

陷阱/异常

内部同步事件导致的上下文切换 (进程中发立的错误)
(中断和异常使用相同的方法解决, 即都进入 kernel mode 后软件处理)

System call 会将当前 user process 导向 kernel process, 完成 system call 后会返回 user process.

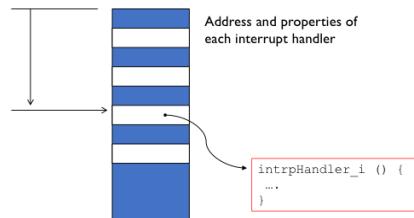
系统调用接口维持着一个数字 → 系统调用的表。

进程控制
文件管理
设备管理
信息保持
通信
保护

Exception & Interrupt

- Exceptions (synchronous) react to an abnormal condition
 - E.g., Map the swapped-out page back to memory
 - Divide by zero
 - Illegal memory accesses
- Interrupts (asynchronous) preempt normal execution
 - Notification from device (e.g., new packets, disk I/O completed)
 - Preemptive scheduling (e.g., timer ticks)
 - Notification from another CPU (i.e., Inter-processor Interrupts)

- Same procedure
 - Stop execution of the current program
 - Start execution of a handler
 - Processor accesses the handler through an entry in the Interrupt Descriptor Table (IDT)
 - Each interrupt is defined by a number



异常是同步的，针对非正常情况

中断是异步的，针对正常执行

OS 维持着 IDT

IDT 的每一项都是指向 handler 的一个指针
发生 Exception 或 interrupt 后 OS 会进入内核态，
CPU 会根据错误码在 IDT 中查表跳转。

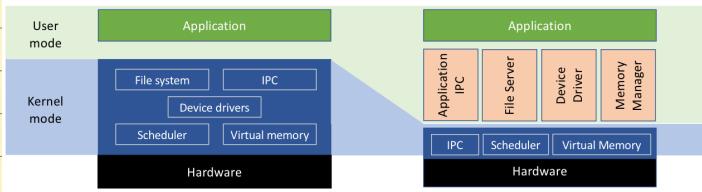
Kernel Structures

Monolithic Kernel 宏内核

- A monolithic kernel is an operating system software framework that holds all privileges to access I/O devices, memory, hardware interrupts and the CPU stack.
- Monolithic kernels contain many components, such as memory subsystems and I/O subsystems, and are usually very large
 - Including filesystems, device drivers, etc.
- Monolithic kernel is the basis for Linux, Unix, MS-DOS.

Micro Kernel 微内核

- Microkernels outsource the traditional operating system functionality to ordinary user processes for better flexibility, security, and fault tolerance.



- OS functionalities are pushed to user-level servers (e.g., user-level memory manager)
- User-level servers are trusted by the kernel (often run as root)
- Protection mechanisms stay in kernel while resource management policies go to the user-level servers
- Representative micro-kernel OS
 - Mach, 1980s at CMU
 - seL4, the first formally verified micro-kernel, <http://sel4.systems/>

内核集成了所有特权操作

尽量使内核小，只有基本功能

将部分功能从内核态移到用户态

因此有的功能的错误不会影响到内核。

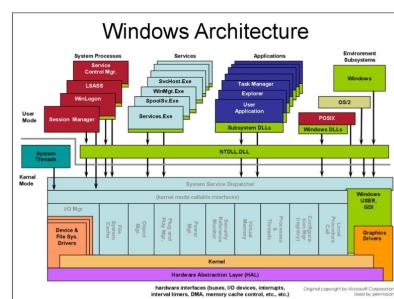
用户层被内核信任。

保护机制仍在内核中，但资源管理移至用户层。

- Pros
 - Kernel is more responsive (kernel functions in preemptible user-level processes)
 - Better stability and security (less code in kernel)
 - Better support of concurrency and distributed OS (later....)
- Cons
 - More IPC needed and thus more context switches (slower)

Hybrid Kernel

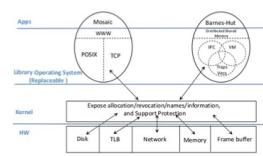
- A combination of a monolithic kernel and a micro kernel
- Example: Windows OS



需要更多的IPC \Rightarrow 更多的上下文切换。

Exokernel 外内核

- An OS concept that separates security from abstraction
 - Kernel is rather small: security multiplexing
 - Library OS run as processes: OS abstraction
- Pros
 - LibOS directly manages resources - efficiency (e.g., database, web server)
 - Easy experimentation of new kernel design
- Cons:
 - Mostly just a concept
 - Several well-known research papers
- Variants: nanokernel, picokernel

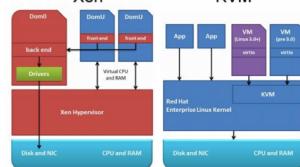


将安全与抽象分开

用 libOS 管理资源

Virtualization & Hypervisors

- Hypervisor (or virtual machine manager/monitor, or VMM) emphasizes on **virtualization** and **isolation**
 - OS can run on hypervisor (almost) without modification
 - Resource partition among VMs
 - Micro kernel and exokernel can sometimes be used to implement hypervisors



Is QEMU a hypervisor? **yes**

hypervisor 强化了虚拟化和隔离。

OS 可运行在 hypervisor 之上。

不同 VM 的资源是隔离的。

微内核和外内核可用于实现 hypervisor

OS Design Principles

- Internal structure of different Operating Systems can vary widely
 - Start by defining goals and specifications
 - Affected by choice of hardware, type of system
- User goals and System goals
 - User goals - operating system should be convenient to use, easy to learn, reliable, safe, and fast
 - System goals - operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient
- OS separates policies and mechanisms
 - Policy: which software could access which resource at what time
 - E.g., if two processes access the same device at the same time, which one goes first
 - E.g., if a process hopes to read from keyboard
 - Mechanism: How is the policy enforced
 - E.g., request queues for devices, running queues for CPUs
 - E.g., access control list for files, devices, etc.
 - The separation of policy from mechanism is a very important principle, it allows maximum flexibility if policy decisions are to be changed later

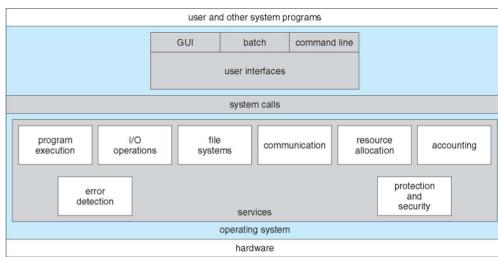
OS 要分离策略和机制

策略: 哪个软件能在何时访问哪种资源。

机制: 策略如何进行。

Operating System Services

- Operating system provides a set of services to application programs



服务是给应用程序提供的

- One set of operating-system services provides functions that are helpful to the user:

- **User interface** - Almost all operating systems have a user interface (UI)
 - Varies between Command-Line (CLI), Graphics User Interface (GUI), Batch
- **Program execution** - The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error)
- **I/O operations** - A running program may require I/O, which may involve a file or an I/O device
- **File-system manipulation** - The file system is of particular interest. Obviously, programs need to read and write files and directories, create and delete them, search them, list file information, permission management
- **Communications** - Processes may exchange information, on the same computer or between computers over a network
 - Communications may be via shared memory or through message passing (packets moved by the OS)
- **Error detection** - OS needs to be constantly aware of possible errors
 - May occur in the CPU and memory hardware, in I/O devices, in user program
 - For each type of error, OS should take the appropriate action to ensure correct and consistent computing
 - Debugging facilities can greatly enhance the user's and programmer's abilities to efficiently use the system

- Another set of OS functions exists for ensuring the efficient operation of the system itself via resource sharing

- **Resource allocation** - When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
 - Many types of resources - Some (such as CPU cycles, main memory, and file storage) may have special allocation code, others (such as I/O devices) may have general request and release code
- **Accounting** - To keep track of which users use how much and what kinds of computer resources
- **Protection and security** - The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other
 - Protection involves ensuring that all access to system resources is controlled
 - Security of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts
 - If a system is to be protected and secure, precautions must be instituted throughout it. A chain is only as strong as its weakest link

User Operating System Interface - CLI

- Command Line Interface (CLI) or command interpreter allows direct command entry

- Sometimes implemented in kernel, sometimes by systems program
- Shells: Bourne shell, C Shell, Bourne-Again Shell, Korn Shell
- Primarily fetches a command from user and executes it
 - Sometimes commands built-in, sometimes just names of programs
 - If the latter, adding new features doesn't require shell modification

User Operating System Interface - GUI

- User-friendly desktop metaphor interface
 - Usually mouse, keyboard, and monitor
 - Icons represent files, programs, actions, etc
 - Various mouse buttons over objects in the interface cause various actions: provide information, options, execute function, open directory (known as a folder)
 - Invented at Xerox PARC
- Many systems now include both CLI and GUI interfaces
 - Microsoft Windows is GUI with CLI "command" shell
 - Apple Mac OS X as "Aqua" GUI interface with UNIX kernel underneath and shells available
 - Solaris is CLI with optional GUI interfaces (Java Desktop, KDE)

System Programs

- System programs provide a convenient environment for program development and execution. They can be divided into:

- File manipulation
- Status information
- File modification
- Programming language support
- Program loading and execution
- Communications
- Application programs

- Most users' view of the operation system is defined by system programs

- Provide a convenient environment for program development and execution
 - Some of them are simply user interfaces to system calls; others are considerably more complex

- File management - Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories

- Status information

- Some ask the system for info - date, time, amount of available memory, disk space, number of users
- Others provide detailed performance, logging, and debugging information
- Typically, these programs format and print the output to the terminal or other output devices
- Some systems implement a registry - used to store and retrieve configuration information

- File modification

- Text editors to create and modify files
- Special commands to search contents of files or perform transformations of the text

- Programming-language support - Compilers, assemblers, debuggers and interpreters sometimes provided

- Program loading and execution

- Communications - Provide the mechanism for creating virtual connections among processes, users, and computer systems

- Allow users to send messages to one another's screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another