Chapter 2: Operating-System Structures

- What are the services provided by an OS ?
- What are system calls ?
- What are some common categories of system calls ?
- What are the principles behind OS design & implementation ?
- What are common ways of structuring an OS ?
- How are VMs and OS related ?

Operating System Services

- Operating-system services that are helpful to the user
  - user interface – *almost all* operating systems have a user interface (UI)
    - Command-Line (CLI)
    - Graphics User Interface (GUI)
    - Touch-Screen Interface
  - program execution – load in memory and run a program
    - end execution, either normally or abnormally (indicating error)
  - I/O operations – allow interaction with I/O devices
    - provide efficiency and protection
  - file-system manipulation – provide uniform access to mass storage
    - create, delete, read, write files and directories
    - search, list file Information
    - permission management.

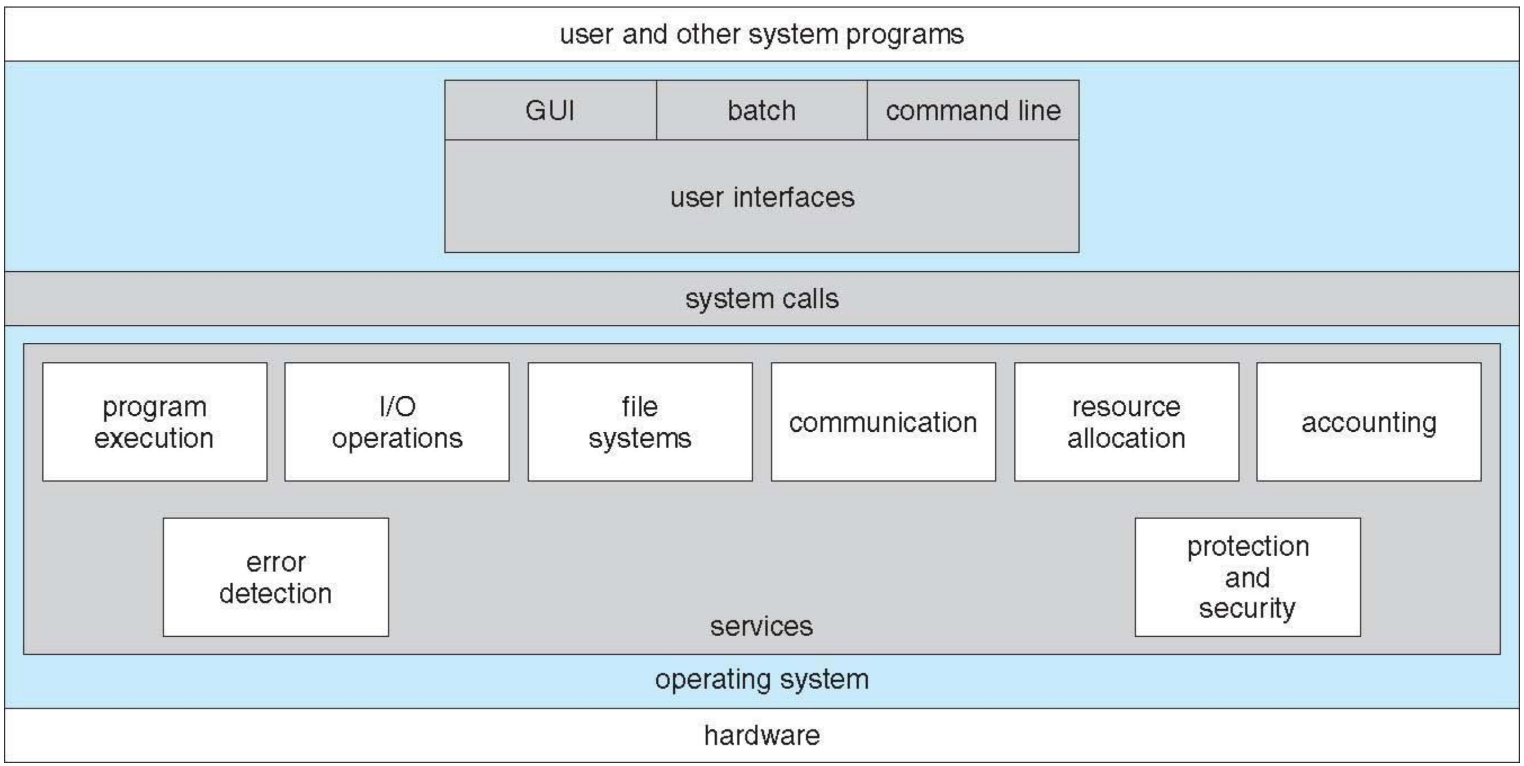
Operating System Services (2)

- Operating-system services that are helpful to the user (cont...)
  - inter-process communication – exchange information among processes
    - shared memory, POSIX *shm_open()*
    - message passing, microkernel OS, RPC, CORBA, etc.
  - error detection – awareness of possible errors
    - CPU and memory hardware (pwer failure, memory fault)
    - I/O device errors (printer out-of-paper, network connection failure)
    - user program (segmentation fault, divide-by-zero)

Operating System Services (3)

- Operating system services for efficient system operation
  - resource allocation – providing access to shared resources in multiuser system
    - CPU cycles, main memory, file storage, I/O devices
  - Accounting – keep track of system resource usage
    - for cost accounting
    - accumulating usage statistics (for profiling, etc.)
  - Protection and security – restrict access to computer resources
    - ensure that all access to system resources is controlled (*protection*)
    - protect system from outsiders (*security*)
    - user authentication, file access control, address space restrictions, etc.

A View of Operating System Services

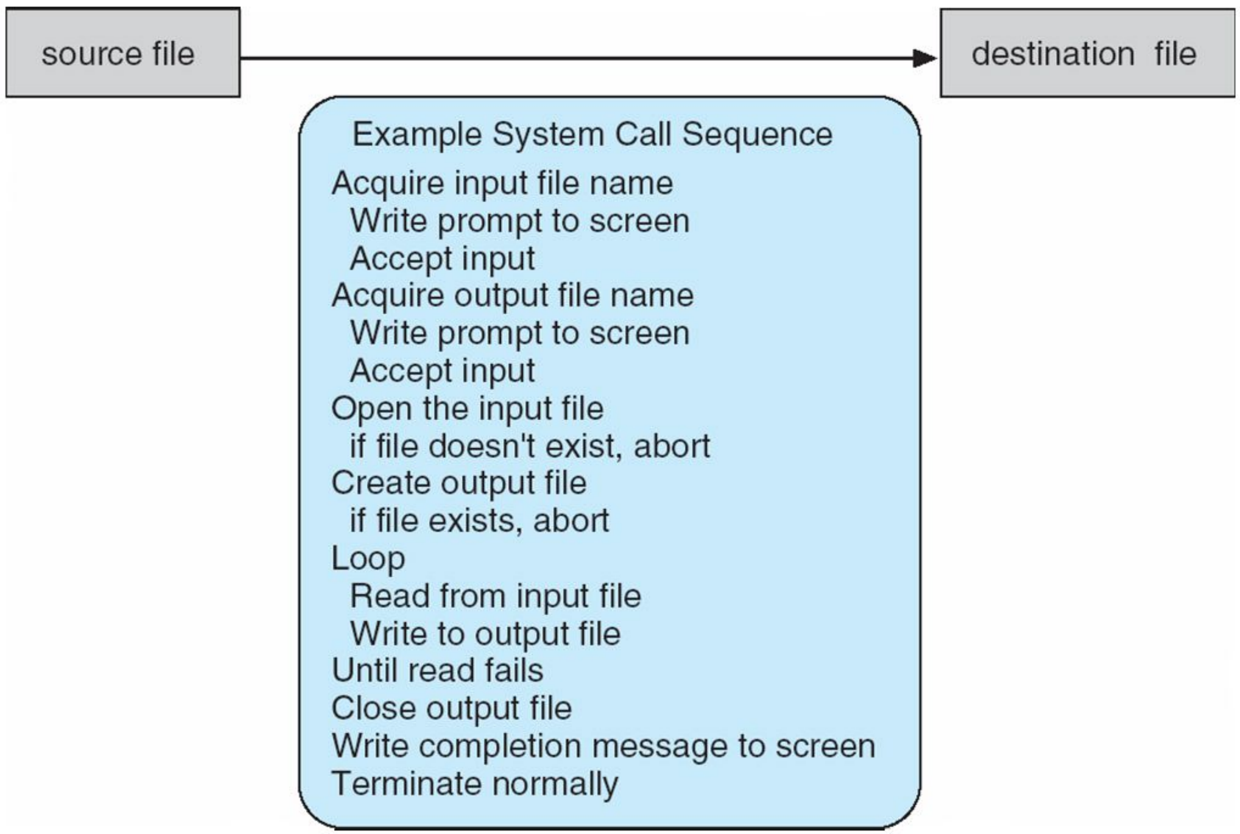


System Calls

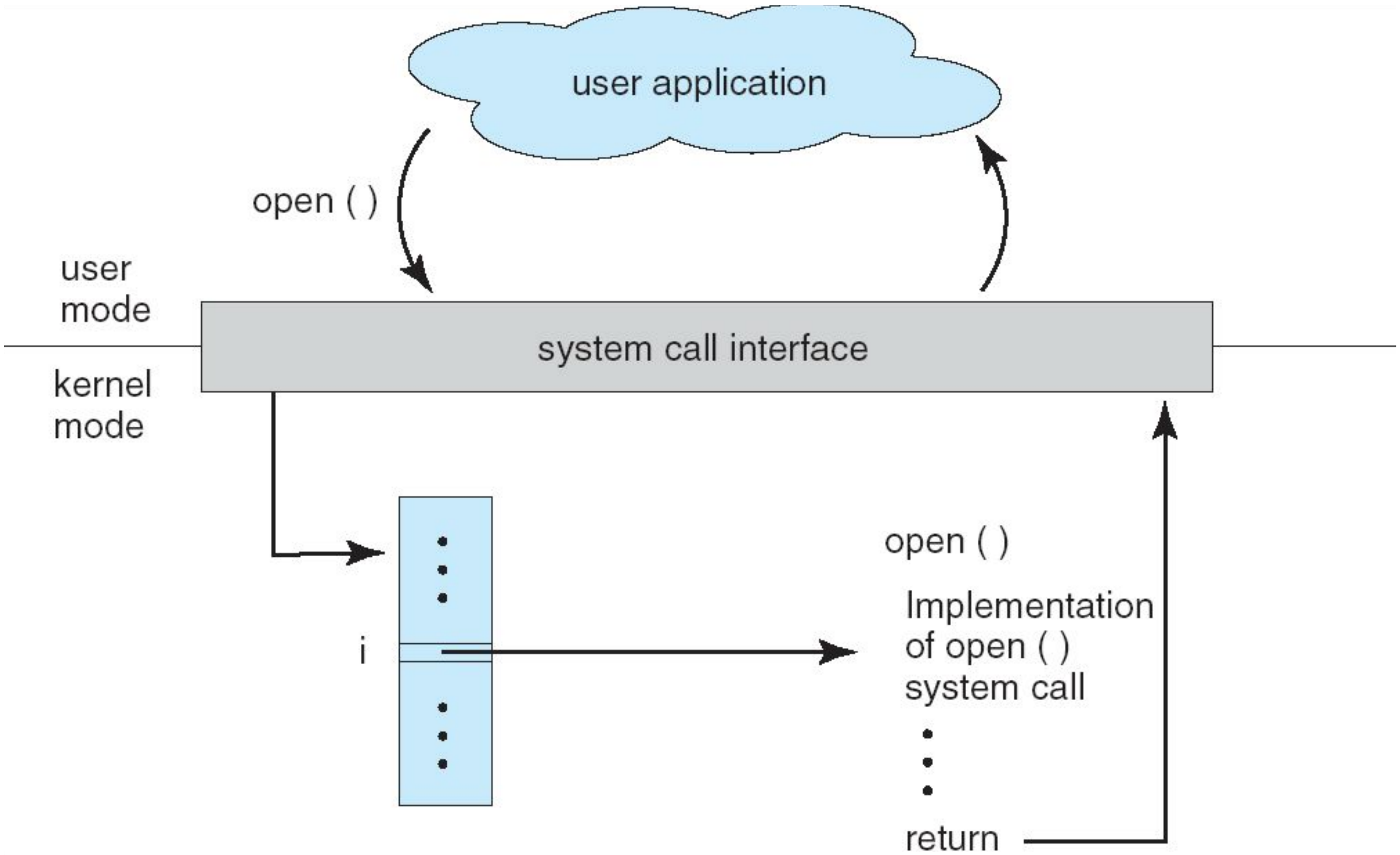
- Programming interface to the services provided by the OS
  - request privileged service from the kernel
  - typically written in a high-level *system* language (C or C++)
- Mostly accessed by programs via a high-level Application Program Interface (API) rather than direct system call use
  - provides a simpler interface to the user than the system call interface
  - reduces coupling between kernel and application, increases portability
- Common APIs
  - Win32 API for Windows
  - POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X)
- Implementation
  - software trap, register contains system call number
  - *syscall* instruction for fast control transfer to the kernel

Example of System Calls

- System call sequence to copy the contents of one file to another file

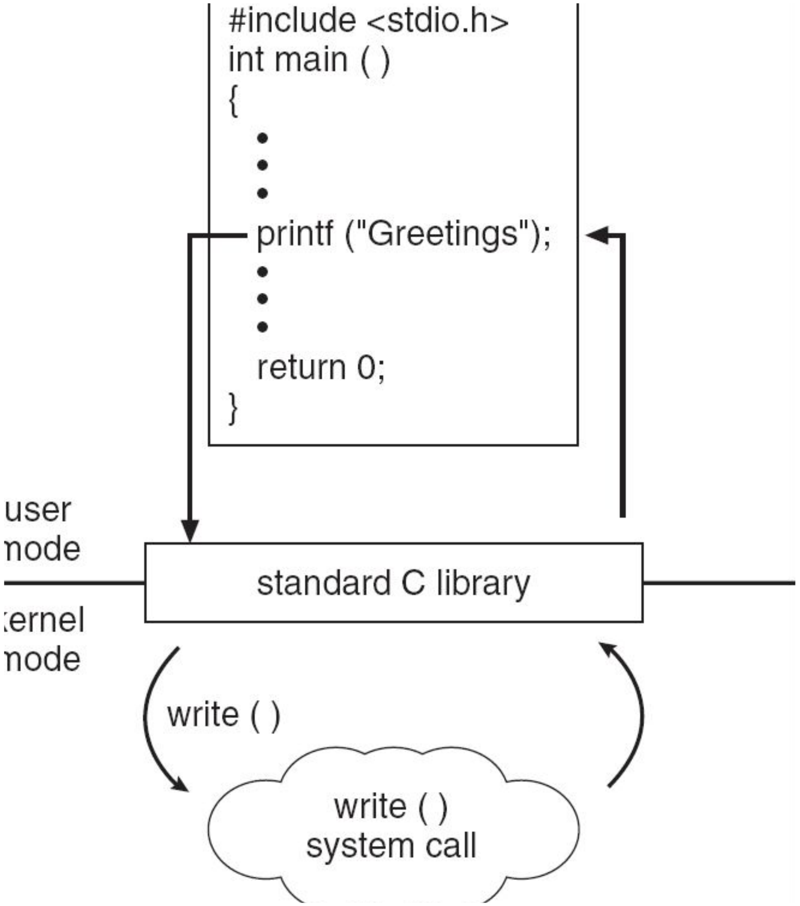


API – System Call – OS Relationship



Standard C Library Example

- C program invoking printf() library call, which calls write() system call



System Call Parameter Passing

- Pass additional information to the system call.
- Three general methods used to pass parameters to the OS
  - pass the parameters in *registers*
    - simplest, fastest
    - what if more parameters than registers ?
  - store arguments in a block on stack
    - pass stack location in a register
  - parameters *pushed* on the *stack* by the program and *popped* off the stack by the operating system
- Pure register method is hardly ever used
  - block and stack methods do not limit the number or length of parameters being passed

Types of System Calls

- Process control
  - create process, terminate process, get/set process attributes, wait event, signal event, allocate and free memory
- File management
  - create, delete, open, close, read, write a file, get/set file attributes
- Device management
  - request, release, read, write, reposition device, get/set device attributes
- Information maintenance
  - get/set time/date, get/set process/file/device attributes
- Communications
  - create/delete connection, send/receive messages
- Protection
  - set/get file/device permissions, allow/deny system resources

Examples of System Calls

	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()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

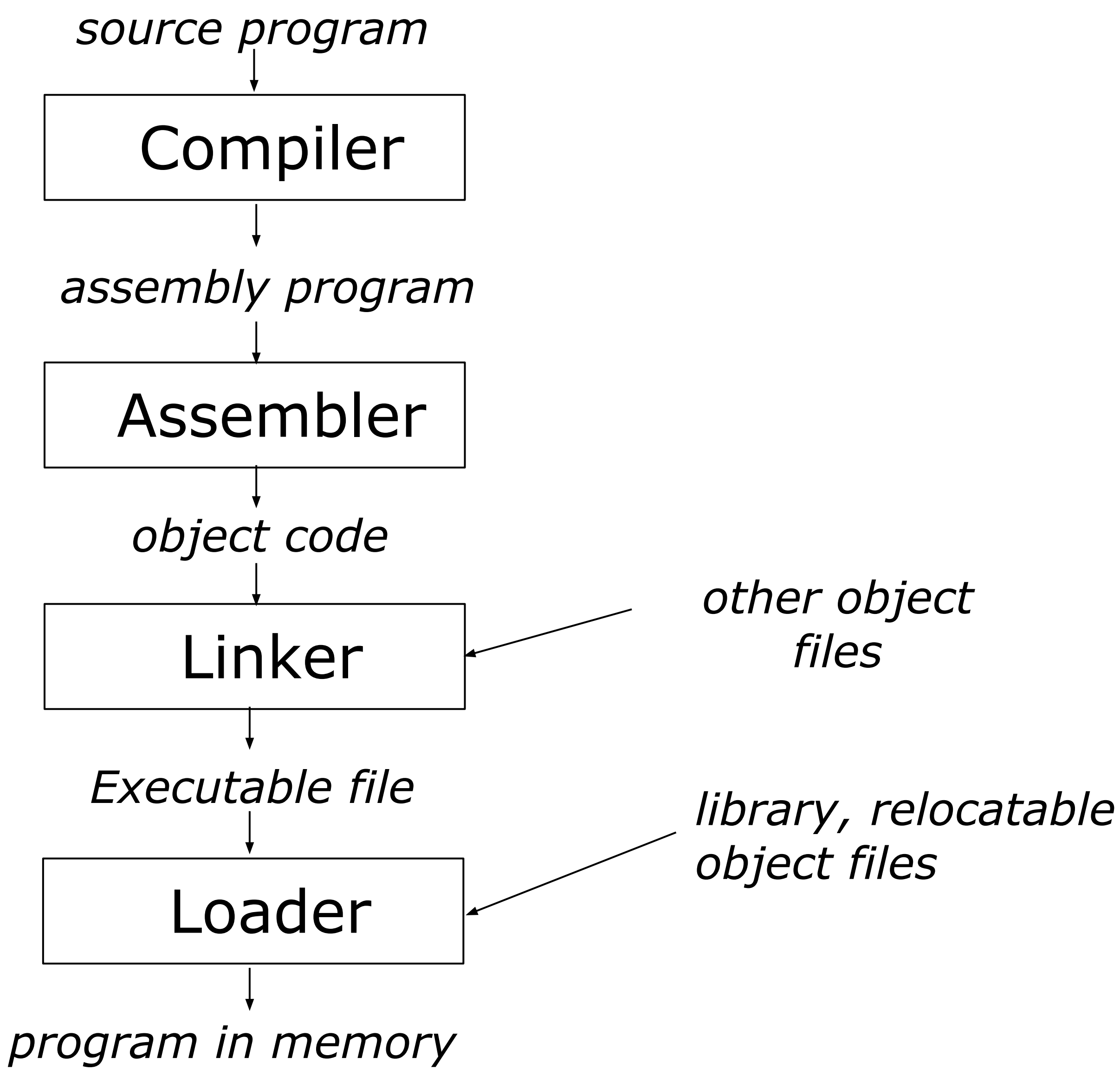
System Programs

- User-level utility programs shipped with the OS
  - ease the job of program development and execution
  - not part of the OS *kernel*
- System programs can be divided into:
  - file manipulation
  - status information
  - file modification
  - programming language support
  - program loading and execution
  - communications
  - application programs

System Programs (2)

- File management
  - mkdir, cp, rm, lpr, ls, ln, etc.
- Status information
  - date, time, ds, df, top, ps, etc.
- File modification
  - editors such as vi and emacs, find, grep, etc.
- Programming language support
  - compilers, assemblers, debuggers, such as gcc, masm, gdb, perl, java, etc.
- Program loading and execution
  - ld
- Communications
  - ssh, mail, write, ftp

Role of Linker and Loader



OS Design and Implementation

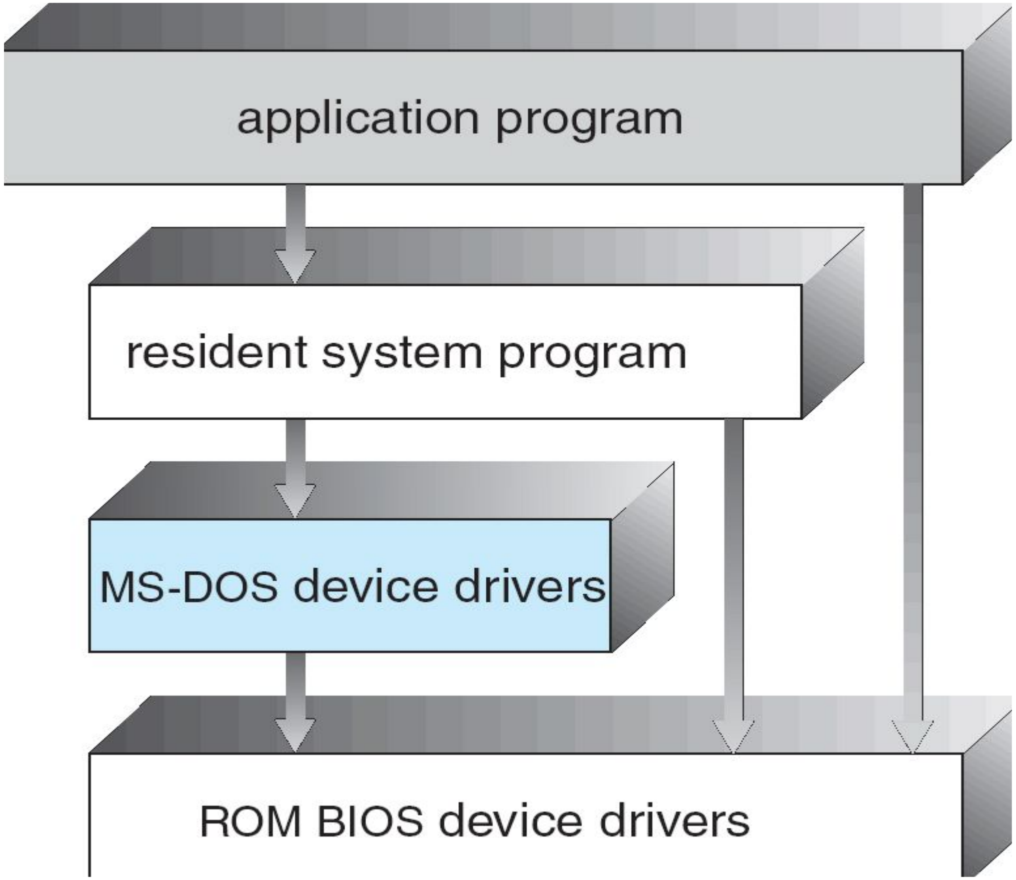
- Design
  - type of system – batch, time-shared, single/multi user, distributed, real-time, embedded
  - user goals – convenience, ease of use and learn, reliable, safe, fast
  - system goals – ease of design, implementation, and maintenance, as well as flexible, reliable, error-free, and efficient
- Mechanism
  - **policy –** what will be done?
  - **mechanism –** how to do it?
- Implementation
  - higher-level language – easier, faster to write, compact, maintainable, easy to debug, portable
  - assembly language – more efficient

Operating System Structure

- Engineering an operating system
  - modularized, maintainable, extensible, etc.
- Simple Structure
  - Characteristics
    - monolithic
    - poor separation between interfaces and levels of functionality
    - ill-suited design, difficult to maintain and extend
  - Reasons
    - growth beyond original scope and vision
    - lack of necessary hardware features during initial design
    - guided more by initial hardware constraints than by sound software engineering principles
    - eg., MS-DOS, UNIX

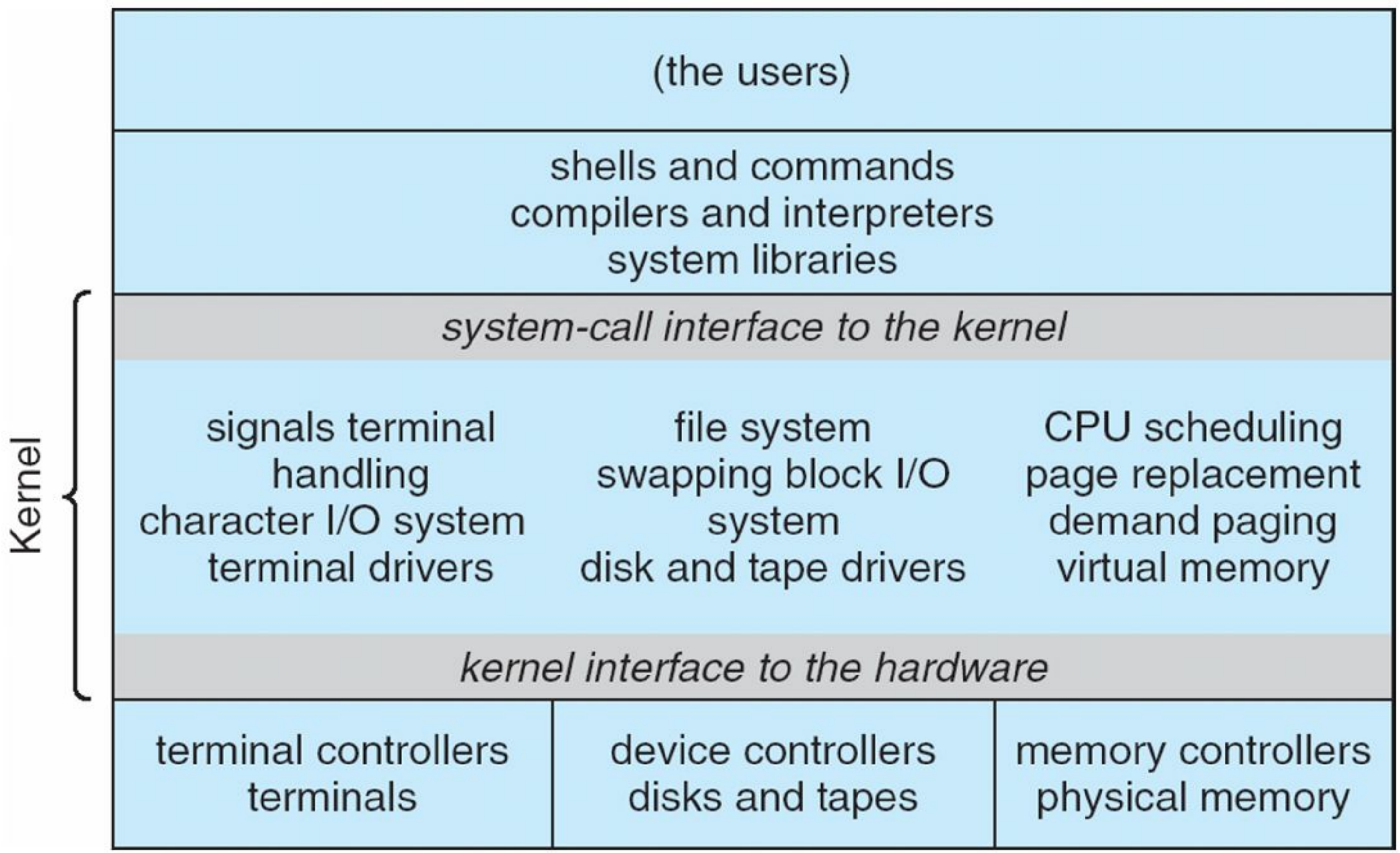
OS Structure - Monolithic

- MS-DOS layer structure:



OS Structure - Monolithic

- Traditional UNIX system structure

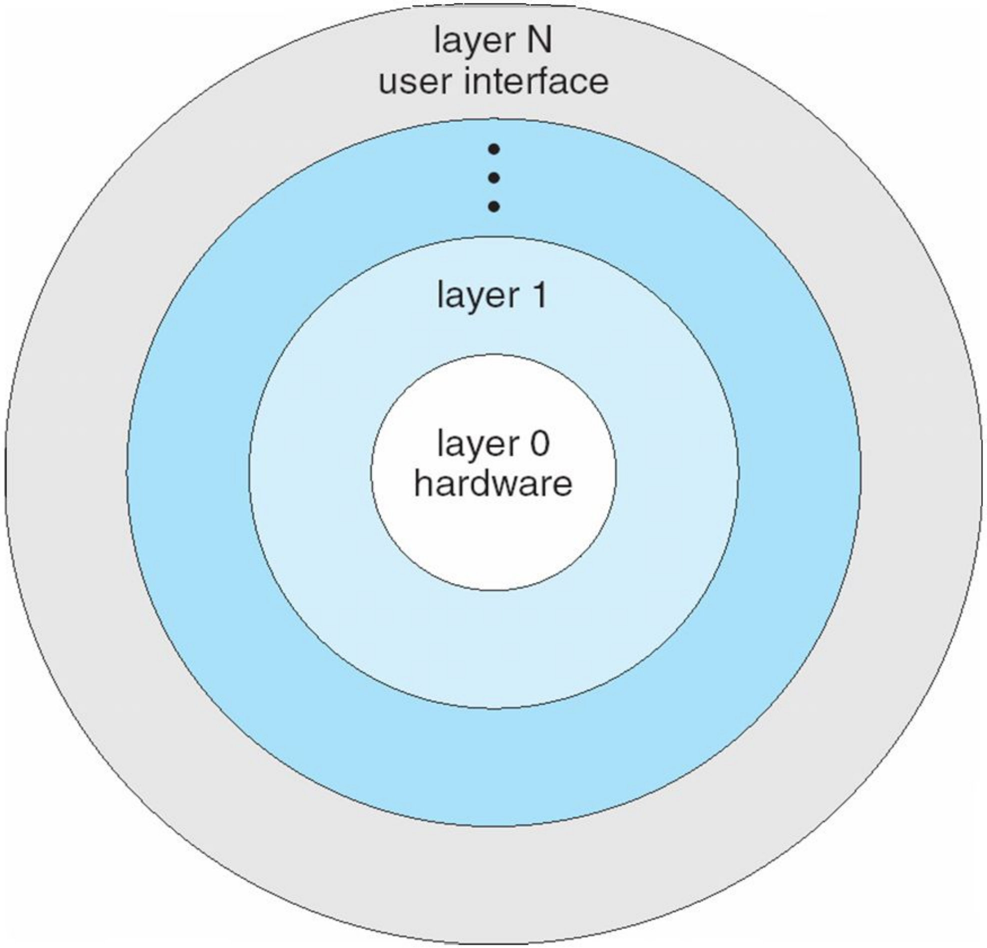


OS Structure - Layered

- Layered approach
  - OS division into a number of layers (levels)
  - upper layers use functions and services provided by lower-level layers
  - Benefits
    - more modular, extensible, and maintainable design
    - achieves information hiding
    - simple construction, debugging, and verification
  - Drawbacks
    - interdependencies make it difficult to cleanly separate functionality among layers
      - eg., backing-store drivers and CPU scheduler
    - less efficient than monolithic designs

OS Structure - Layered

- Layered Operating System

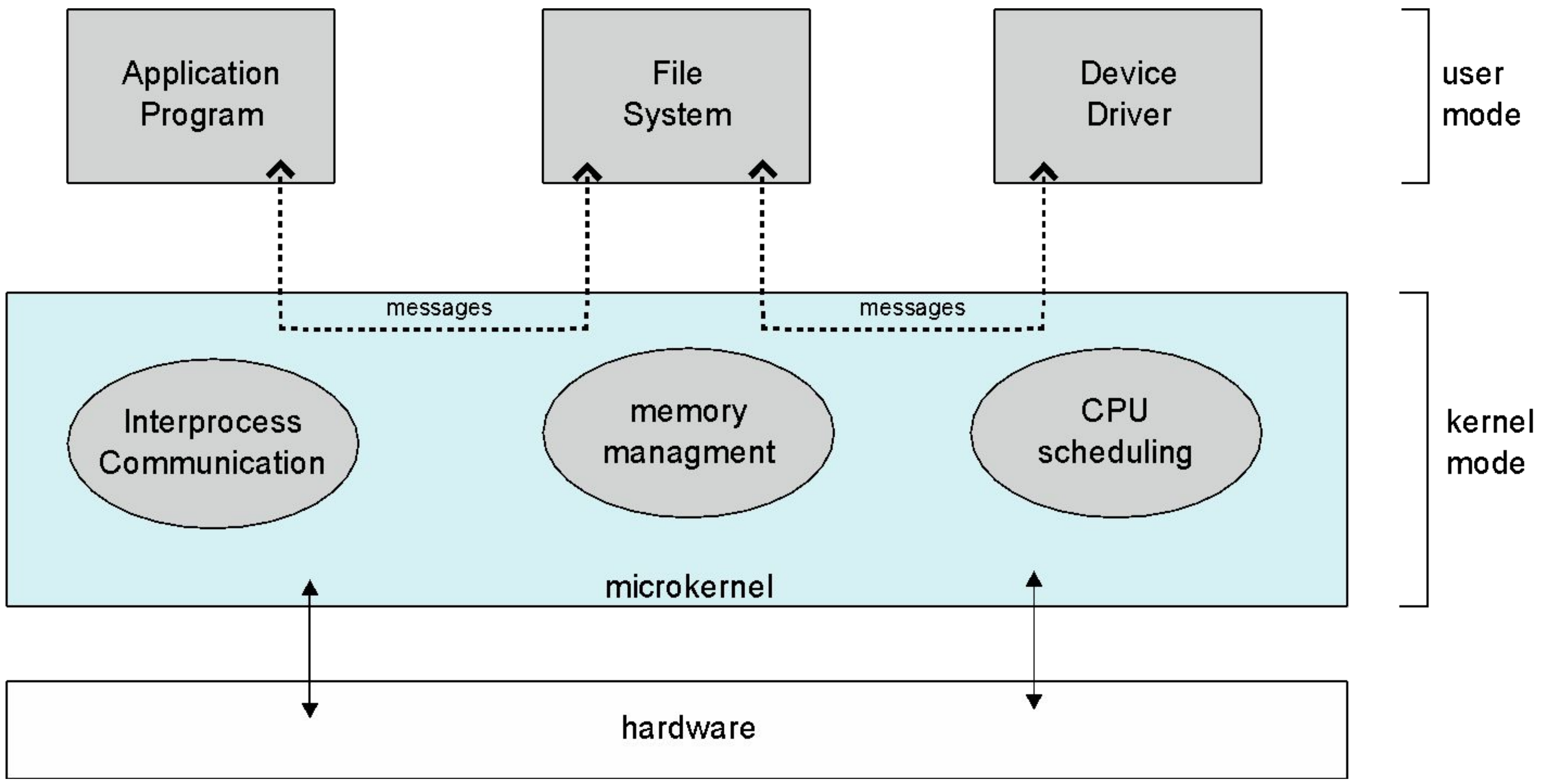


OS Structure - Microkernels

- Microkernel System Structure
  - moves as much functionality from the kernel into “*user*” space
  - communicate between user modules using message passing
  - Benefits
    - easier to extend (user level drivers)
    - easier to port to new architectures
    - more reliable (less code is running in kernel mode)
    - more secure
  - Drawbacks
    - no consensus regarding services that should remain in the kernel
    - performance overhead of user space to kernel space communication

Operating System Structure (7)

- Microkernel system structure

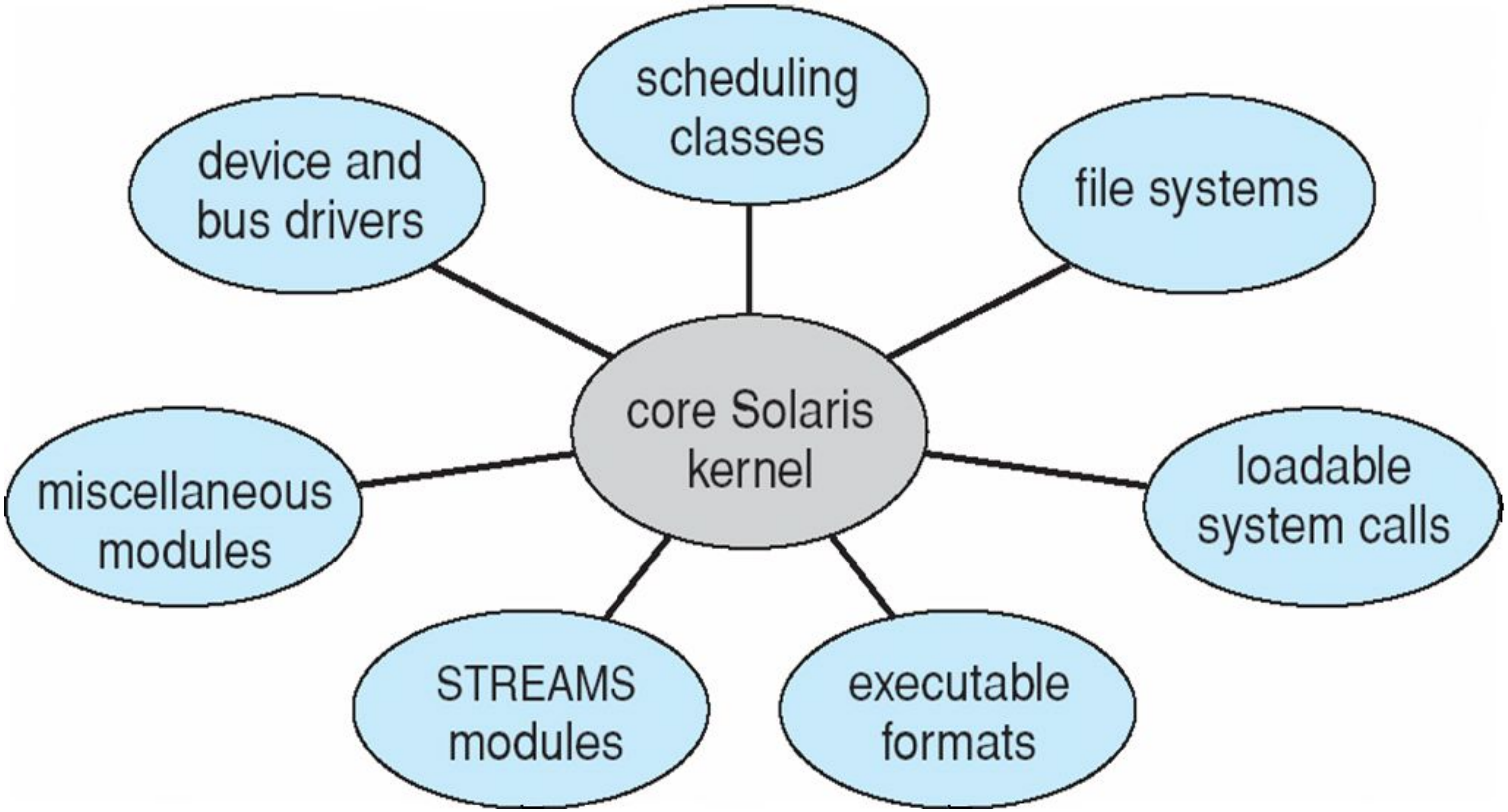


OS Structure - Modules

- Modules
  - uses object-oriented approach
  - kernel provides core functionality, like communications, device drivers
  - additional services are modules linked dynamically
  - services talk directly over interfaces bypassing the kernel
  - Benefits
    - advantages of layered structure but with more flexible
    - advantages of microkernel approach, without message passing overhead
  - Drawbacks
    - not as clean a design as the layered approach
    - not as small a kernel as a microkernel
    - but, achieves best of both worlds as far as possible

OS Structure - Modules

- Solaris modular approach

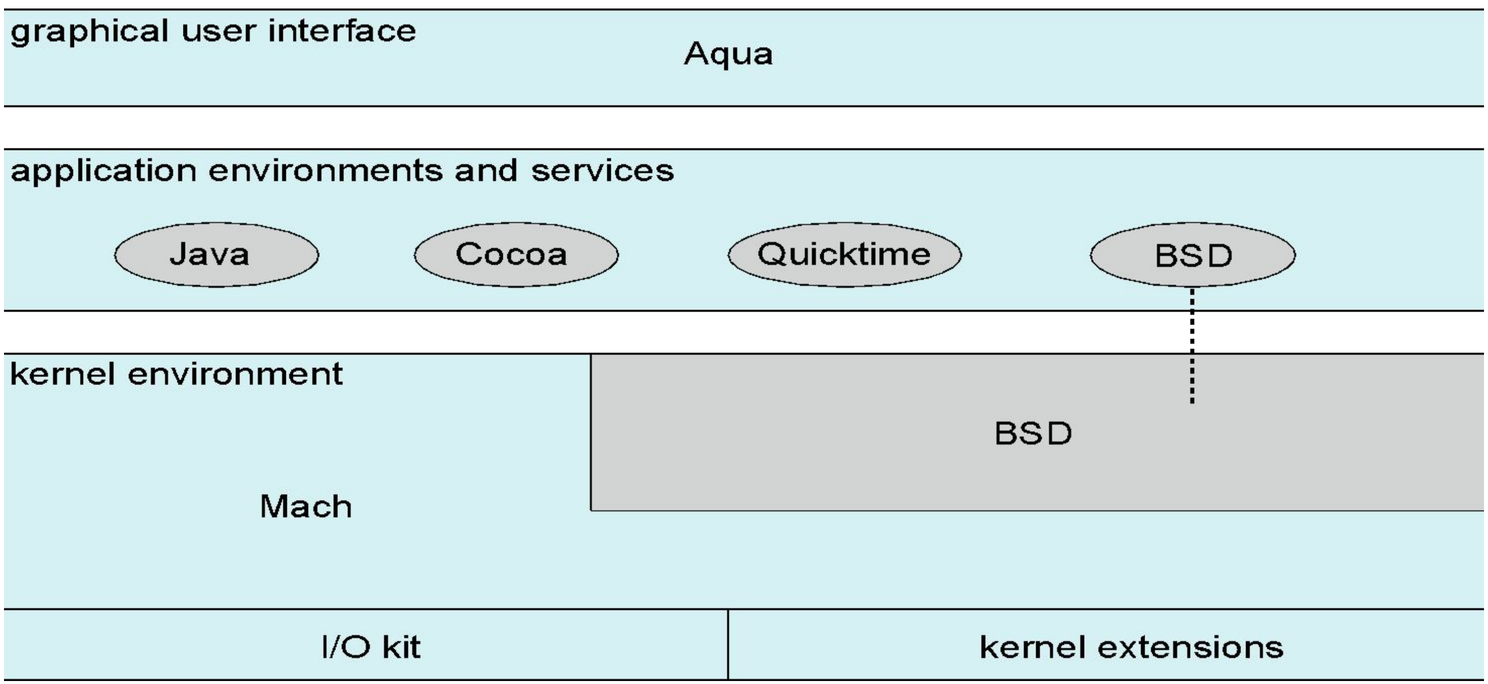


OS Structure – Hybrid Systems

- Hybrid operating systems
  - combine multiple approaches to address performance, security, usability
- Linux
  - Monolithic, since OS is in a single address space
  - Modular, since can be extended dynamically
- Windows
  - Monolithic, but some microkernel aspects
- Hybrid OS – Android OS structure
  - modified Linux kernel for process, memory, device driver management
  - Runtime provided higher-level libraries and ART runtime
  - Uses *bionic*, rather than *glibc*

OS Structure – Hybrid Systems

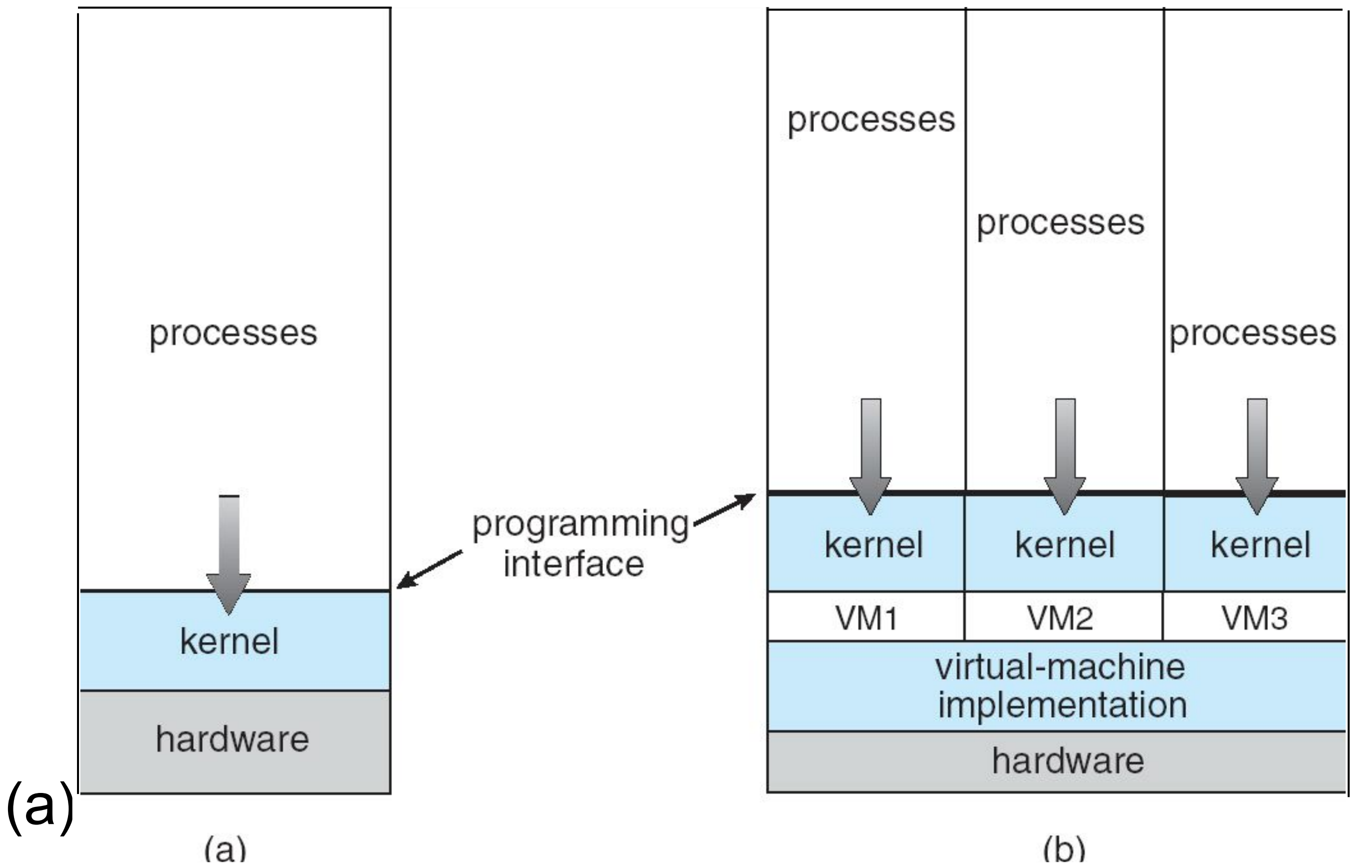
- Example – Apple Mac OS X
  - hybrid, layered
  - Mach microkernel and BSD Unix, plus I/O kit, and dynamically loadable modules for kernel extensions



Virtual Machines

- Generally, exposes a *virtual* interface different from the *physical* real
  - time sharing, multi-user OS as a virtual machine ?
  - abstraction Vs. virtualization ?
- Traditionally, exposes an interface of *some* hardware system
  - includes CPU, memory, disk, network, I/O devices, etc.
  - interface need not be identical to the underlying hardware
- A virtualization layer, called *hypervisor,* takes over control of the host hardware resources
  - creates the illusion that a process has its own computer system
  - each guest provided with a (virtual) copy of underlying computer
  - each guest process can then run another OS and application programs

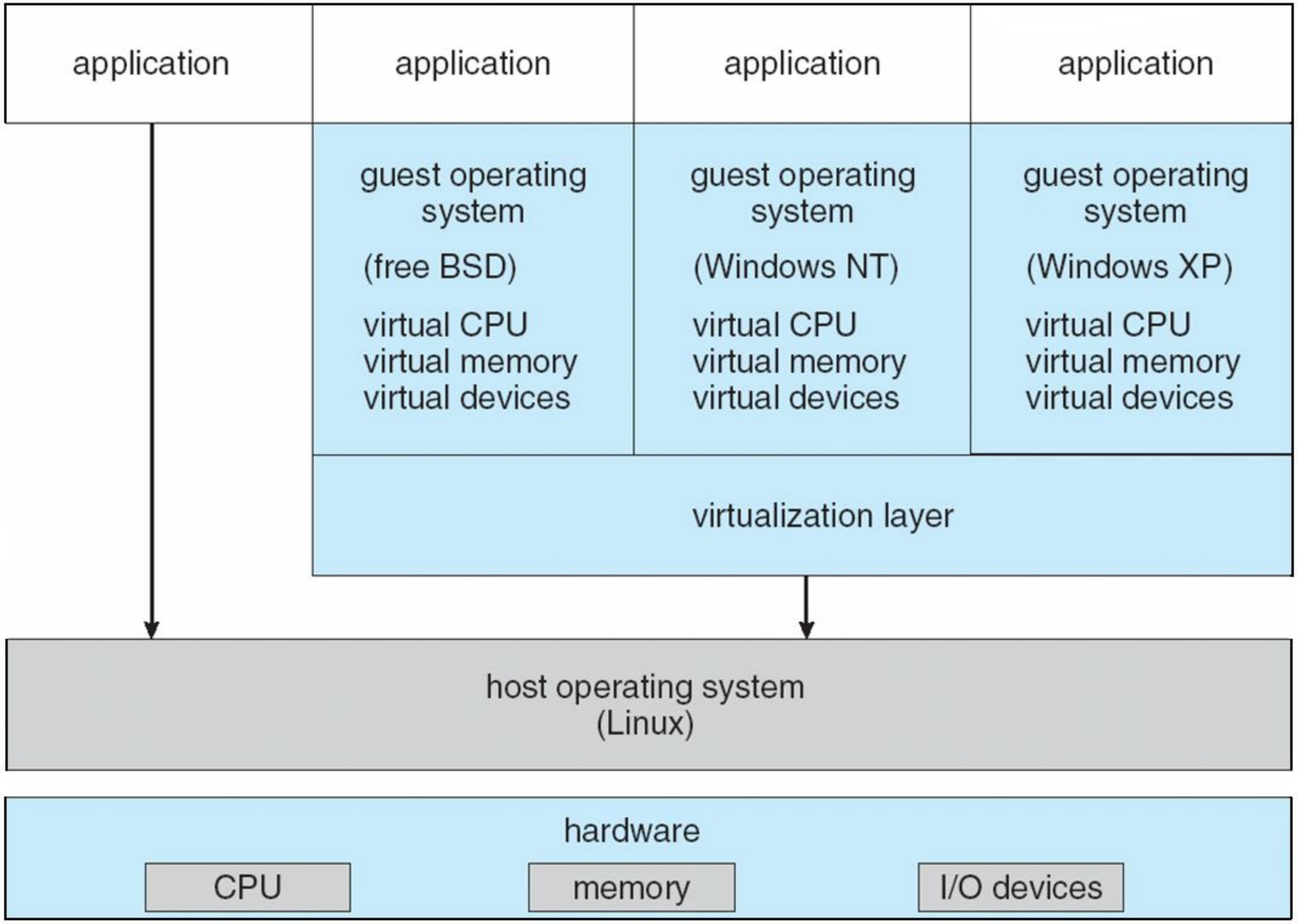
Virtual Machines (2)



Virtual Machines History and Benefits

- History
  - introduced by IBM for their IBM 360/370 line of machines
  - exposed an interface that was identical to the underlying machine
  - ran the single-user, time-sharing CMS operating system on each VM
- Benefits
  - ability to enable multiple execution environments (different operating systems) to share the same hardware
  - application programs in different VMs *isolated* from each other
    - provides protection; can make sharing and communication difficult
  - useful for development, testing (particularly OS)
  - testing cross-platform compatibility
  - consolidation of many low-resource use systems onto fewer busier systems
  - process virtual machines (Java) provide application portability

VMware Architecture



The Java Virtual Machine