CSMC 412

Operating Systems

Prof. Ashok K Agrawala

© 2006 Ashok Agrawala

Set 2

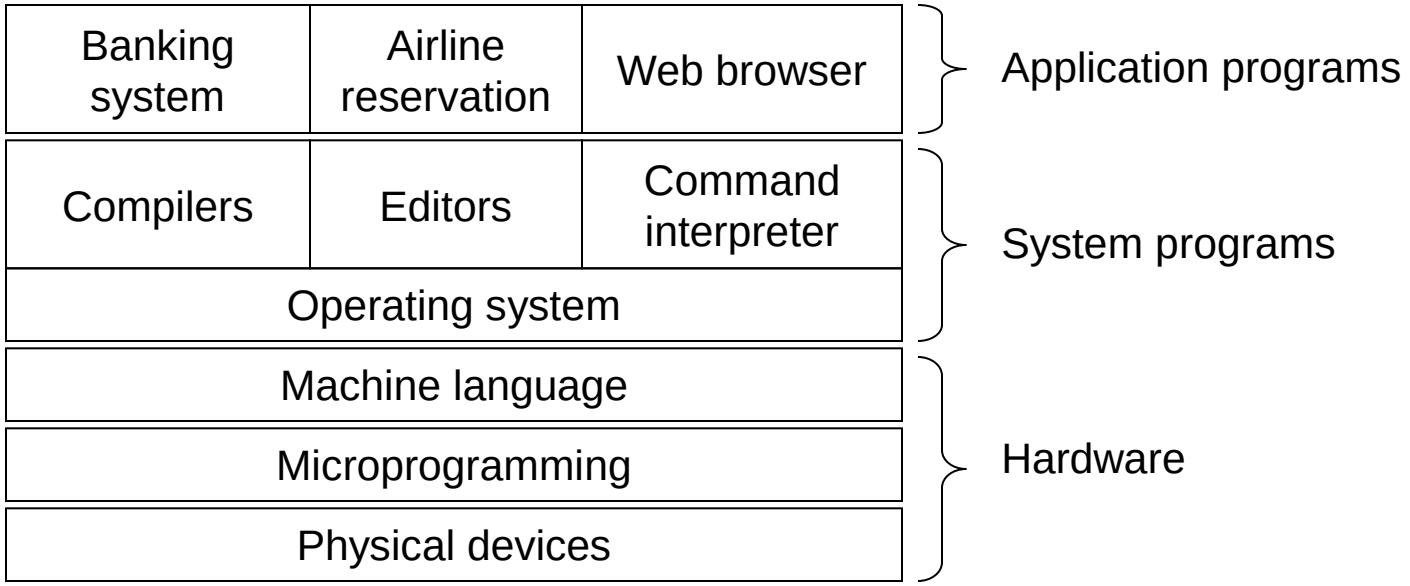
The Evolution of Operating Systems

Topics

- History of Operating Systems
- Tasks of an Operating System
- OS as extension of the hardware
- Main concepts: processes, files, system calls
- Operating system structuring

Operating Systems Concepts

- System software manages resources
- OS hides complexity of underlying hardware
- Layered architectures



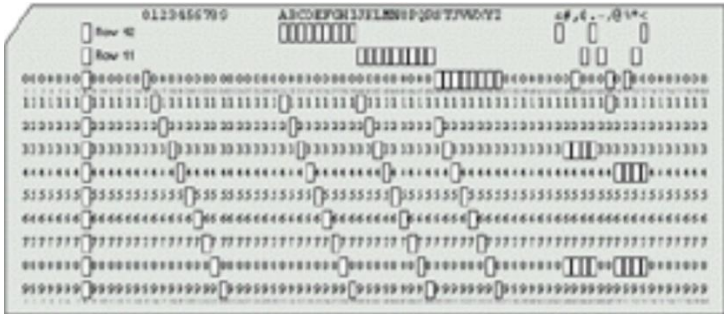
History of operating systems

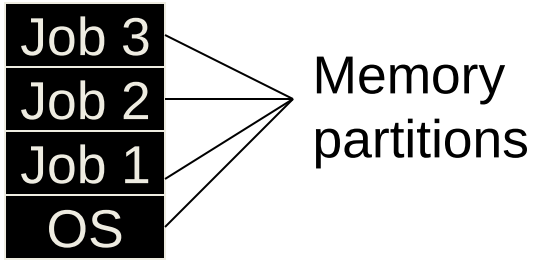
- Batch processing

The elements of the basic IBM 1401 system are the 1401 Processing Unit, 1402 Card Read-Punch, and 1403 Printer.

- Punching cards

Multiprocessing programming





The Evolution of Operating System Functionality

- Batch Job Processing
  - Linkage of library routines to programs
  - Management of files, I/O devices, secondary storage
- Multiprogramming
  - Resource managment and sharing for multiple programs
  - Quasi-simultaneous program execution
  - Single user
- Multiuser/Timesharing Systems
  - Management of multiple simultaneous users interconnected via terminals
  - Fair resource management: CPU scheduling, spooling, mutual exclusion
- Real-Time Systems (process control systems)
  - Management of time-critical processes
  - High requirements with respect to reliability and availability

Tasks of an Operating System

- Processor management - Scheduling
  - Fairness
  - Non-blocking behavior
  - Priorities
- Memory management
  - Virtual versus physical memory, memory hierarchy
  - Protection of competing/conurrent programs
- Storage management – File system
  - Access to external storage media
- Device management
  - Hiding of hardware dependencies
  - Management of concurrent accesses
- Batch processing
  - Definition of an execution order; throughput maximization

Kernel- and User Mode Programs

Typical functionality implemented in either mode:

Kernel:

- Privileged mode
- Strict assumptions about reliability/security of code
- Memory resident
  - CPU-, memory-, Input/Output managment
  - Multiprocessor management, diagnosis, test
  - Parts of file system and of the networking interface

User Space:

- More flexible
- Simpler maintenance and debugging
  - Compiler, assembler, interpreter, linker/loader
  - File system management, telecommunication, network management
  - Editors, spreadsheets, user applications

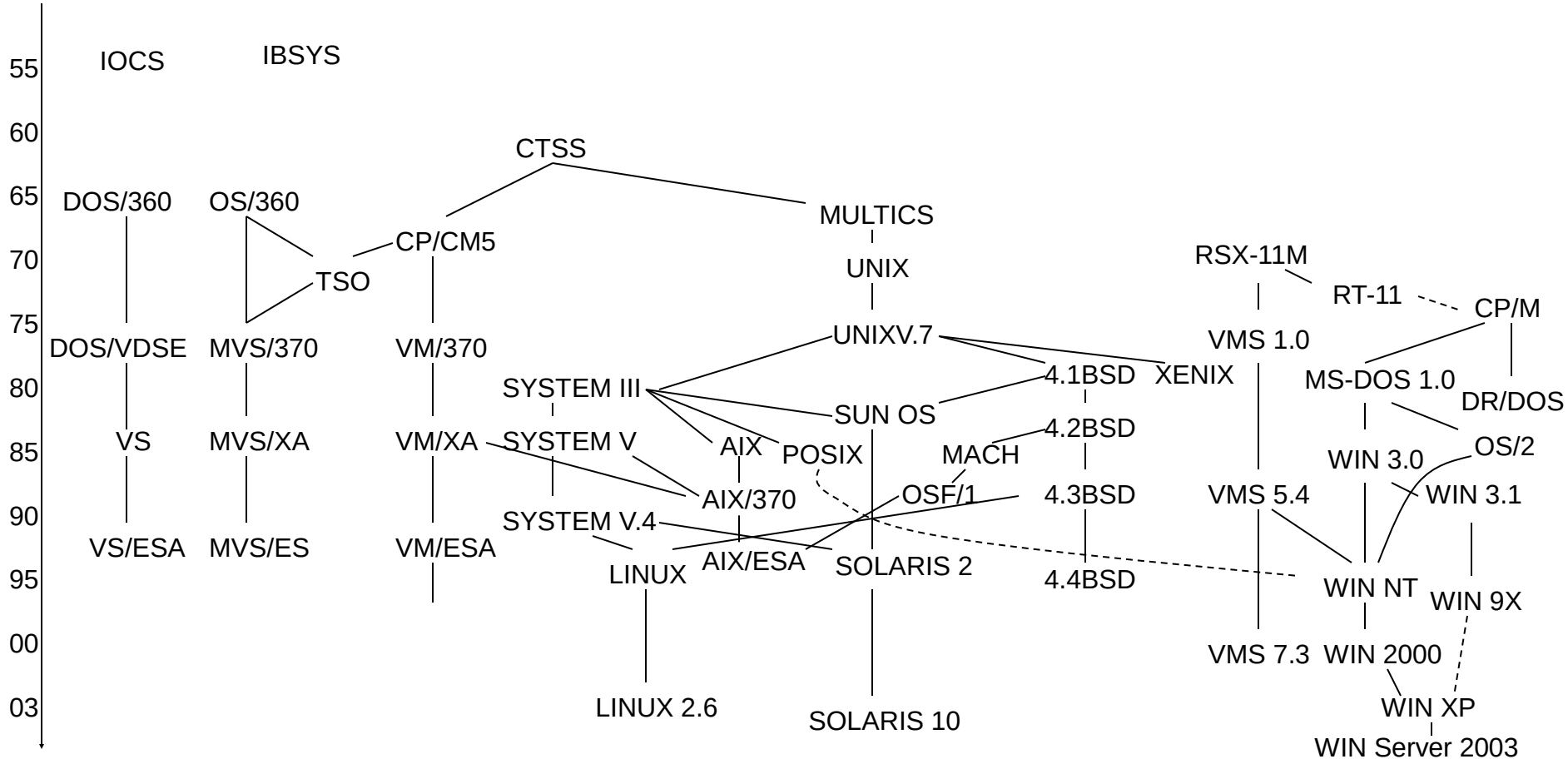
Layered Model of Operating System Concepts

nr	name	typical objects	typical operations
1	Integrated circuits	register, gate, bus	Nand, Nor, Exor
2	Machine language	instruction counter, ALU	Add, Move, Load, Store
3	Subroutine linkage	procedure block	Stack Call, JSR, RTS
4	Interrupts	interrupt handlers	Bus error, Reset
5	Simple processes	process, semaphore	wait, ready, execute
6	Local memory	data block, I/O channel	read, write, open, close
7	Virtual model	page, frame	read, write, swap
8	Process communication	channel (pipe), message	read, write, open
9	File management	files	read, write, open, copy
10	Device management	ext.memory, terminals	read, write
11	I/O data streams	data streams	open, close, read, write
12	User processes	user processes	login, logout, fork
13	Directory management	internal tables	create, delete, modify
14	Graphical user interface	window, menu, icon	OS system calls

OS acts as Extension of Hardware

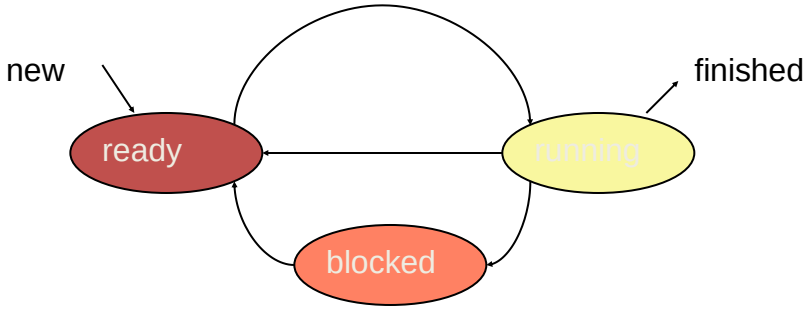
- System view: layered model of OS
  - Implementation details on one layer are hidden from higher layers
- Same machine, different operating systems:
  - IBM PC: DOS, Linux, NeXTSTEP, Windows, SCO Unix
  - DEC VAX: VMS, Ultrix-32, 4.3 BSD UNIX
- Same OS, different machines: UNIX
  - PC (XENIX 286, APPLE A/UX)
  - CRAY-Y/MP (UNICOS - AT&T Sys V)
  - IBM 360/370 (Amdahl UNIX UTS/580, IBM UNIX AIX/ESA)
- Windows NT, XP, 2000, 2003
  - Intel i386 (i486 an NT 4.0), Alpha, PowerPC, MIPS, Itanium

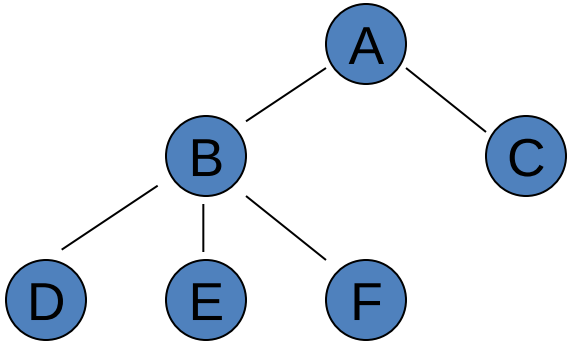
Operating Systems Evolution



Main Concepts: processes

- Processes, process table, core image
- Command interpreter, shell
- Child processes



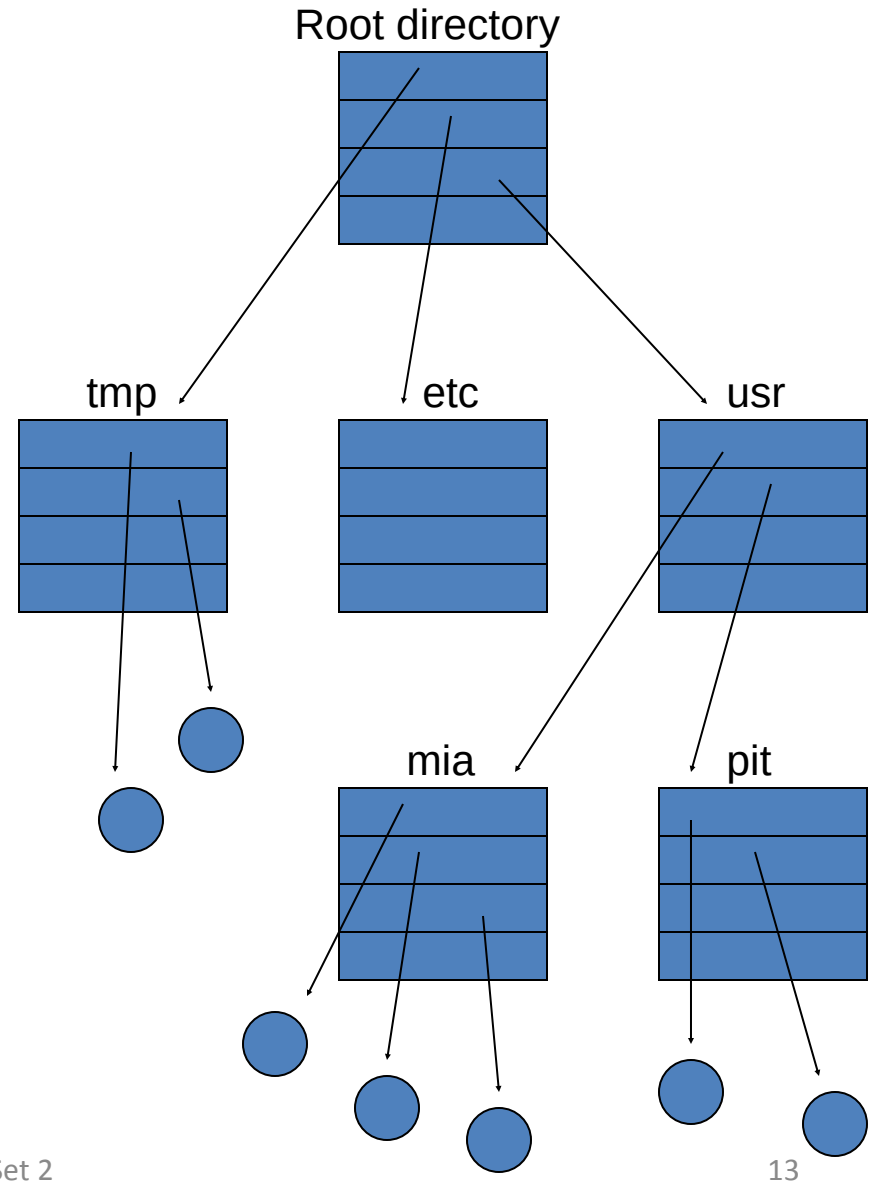


Process tree

- Scheduling, signals
- User identification, group identification

Main Concepts: Files

- Files, directories, root
- Path, working directory
- Protection, rwx bits
- File descriptor, handle
- Special files, I/O devices
- Block I/O, character I/O
- Standard input/output/error
- pipes



Main concepts: system calls

- User programs access operating system services via system calls
- Parameter transmission via trap, register, stack

  count=read(file, buffer, nbytes);

- 5 general classes of system calls:
  - Process control
  - File manipulation
  - Device manipulation
  - Information maintenance
  - communications

Main concepts: shell

- Command interpreter
- Displays prompt, implements input/output redirection
- Background processes, job control, pseudo terminals

$ date

$ date >file

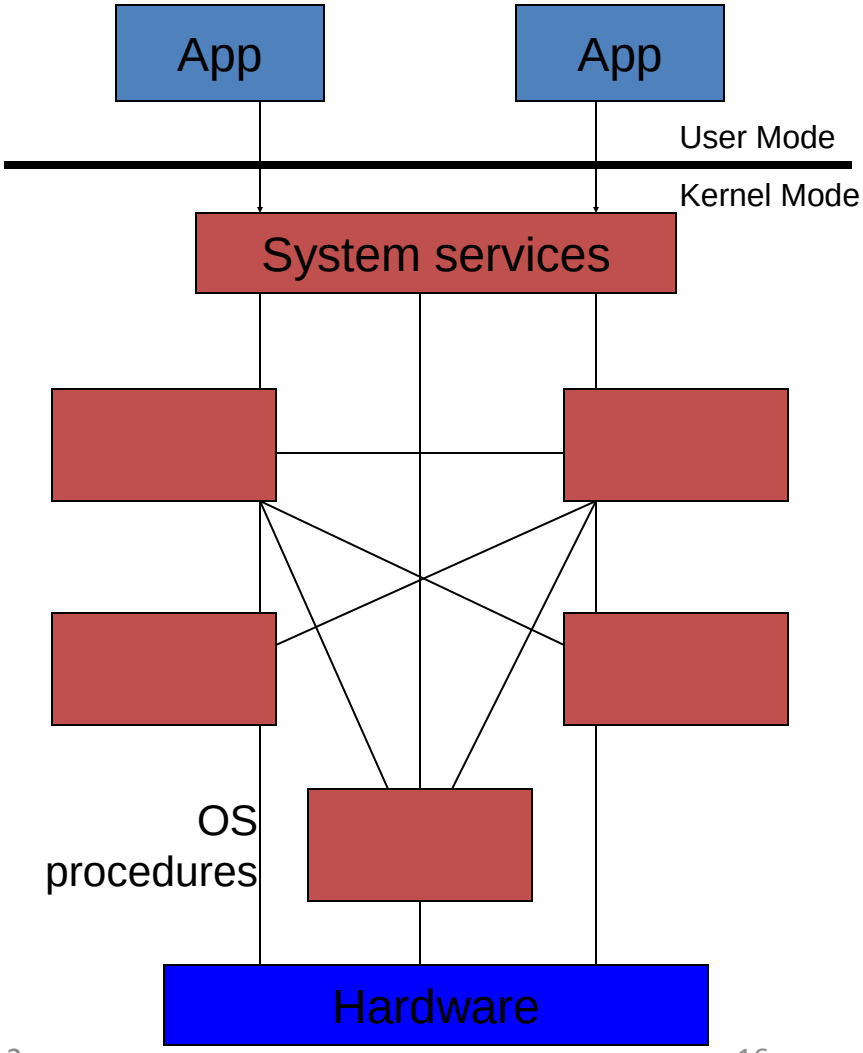
$ sort <file1 >file2

$ cat file1 file2 file3 > /dev/lp1

$ make all >log 2>&1 &

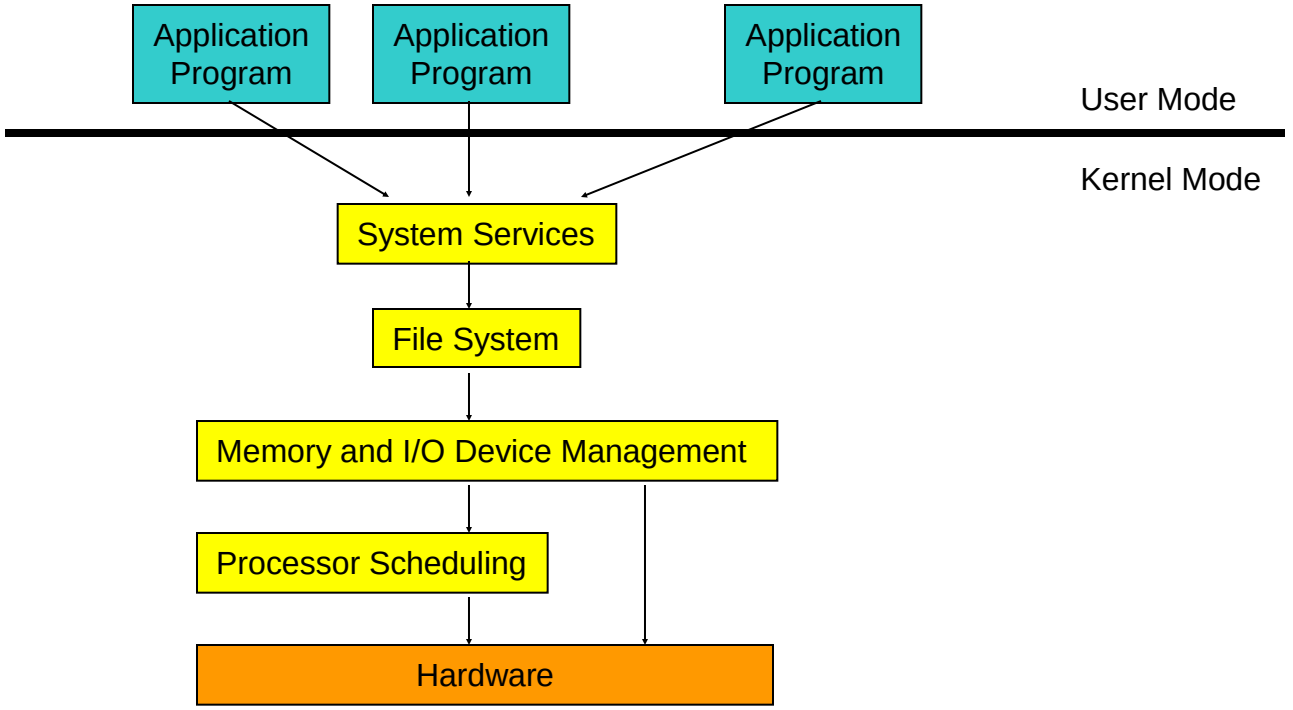
Structuring of Operating Systems

- **Monolithical systems**
- Unstructured
- Supervisor call changes from user mode into kernel mode



Layered OS

- Each layer is given access only to lower-level interfaces

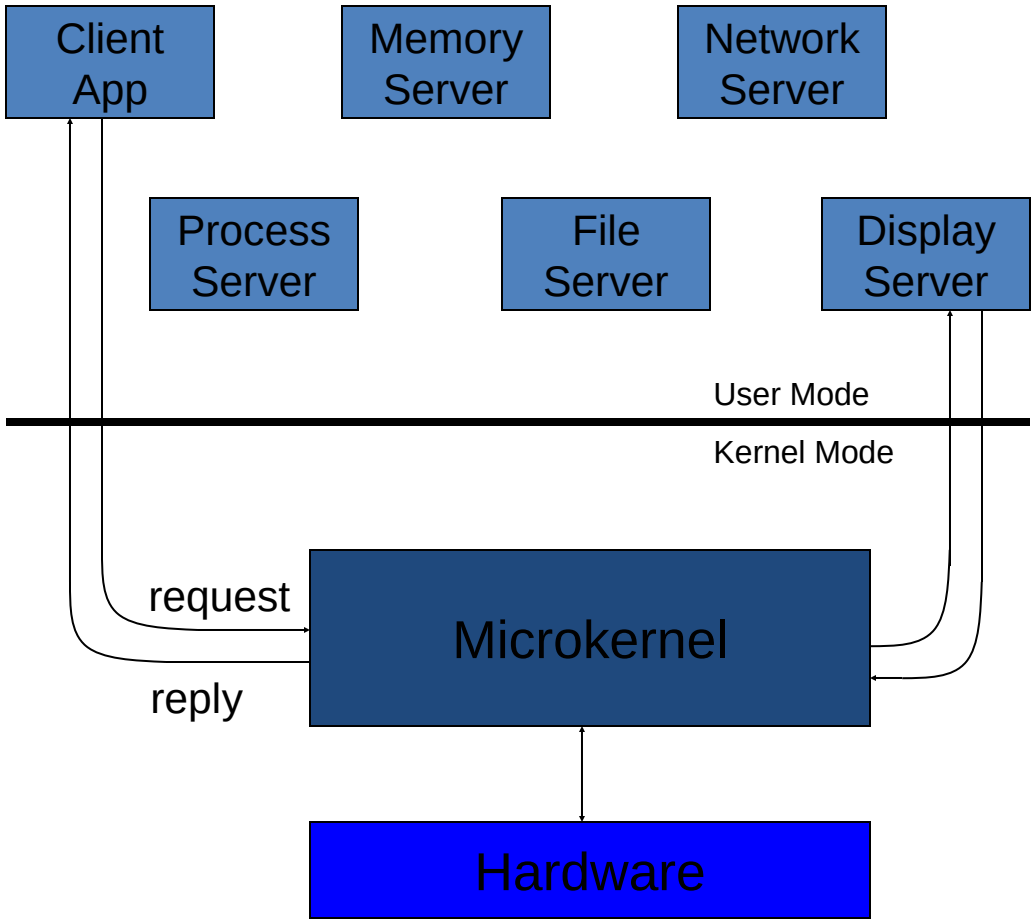


Microkernel OS (Client/server OS)

Kernel implements:

- Scheduling
- Memory Management
- Interprocess communication (IPC)

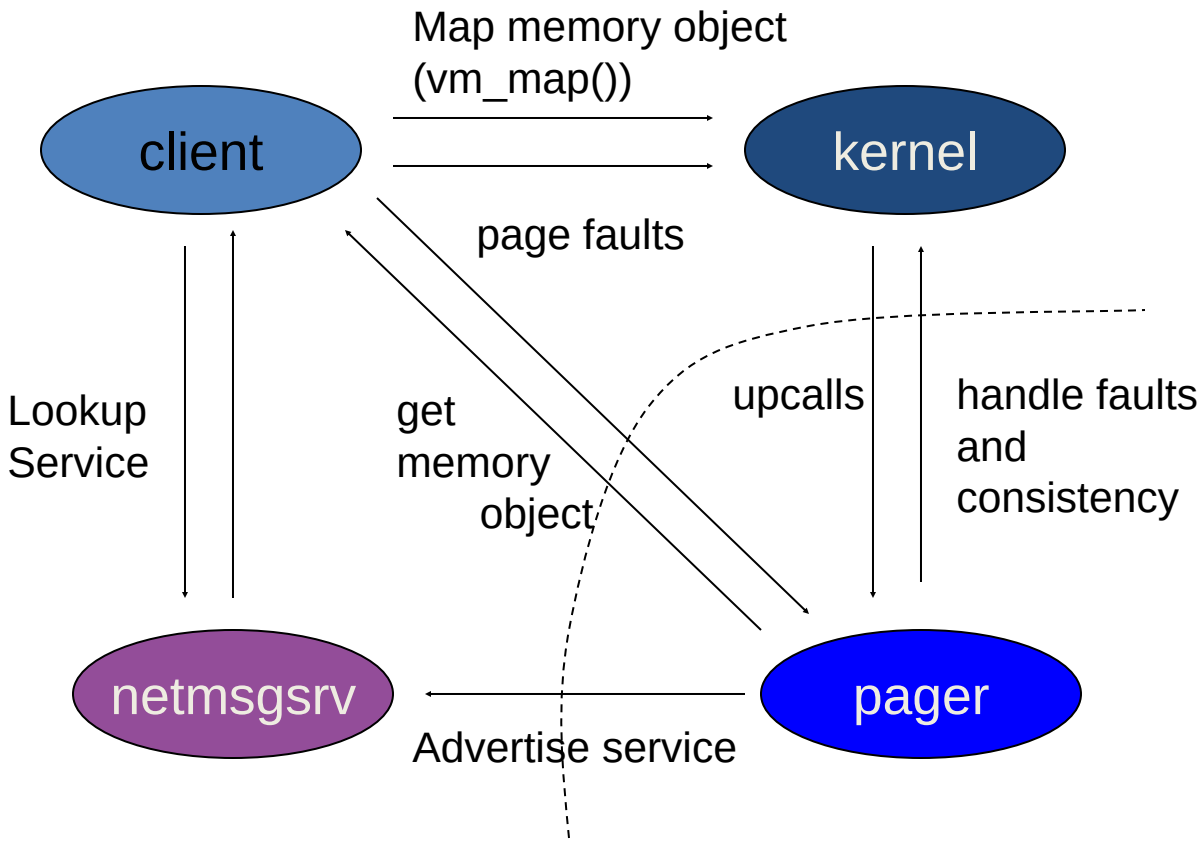
User-mode servers



Mach Microkernel OS
Extended Memory Managment

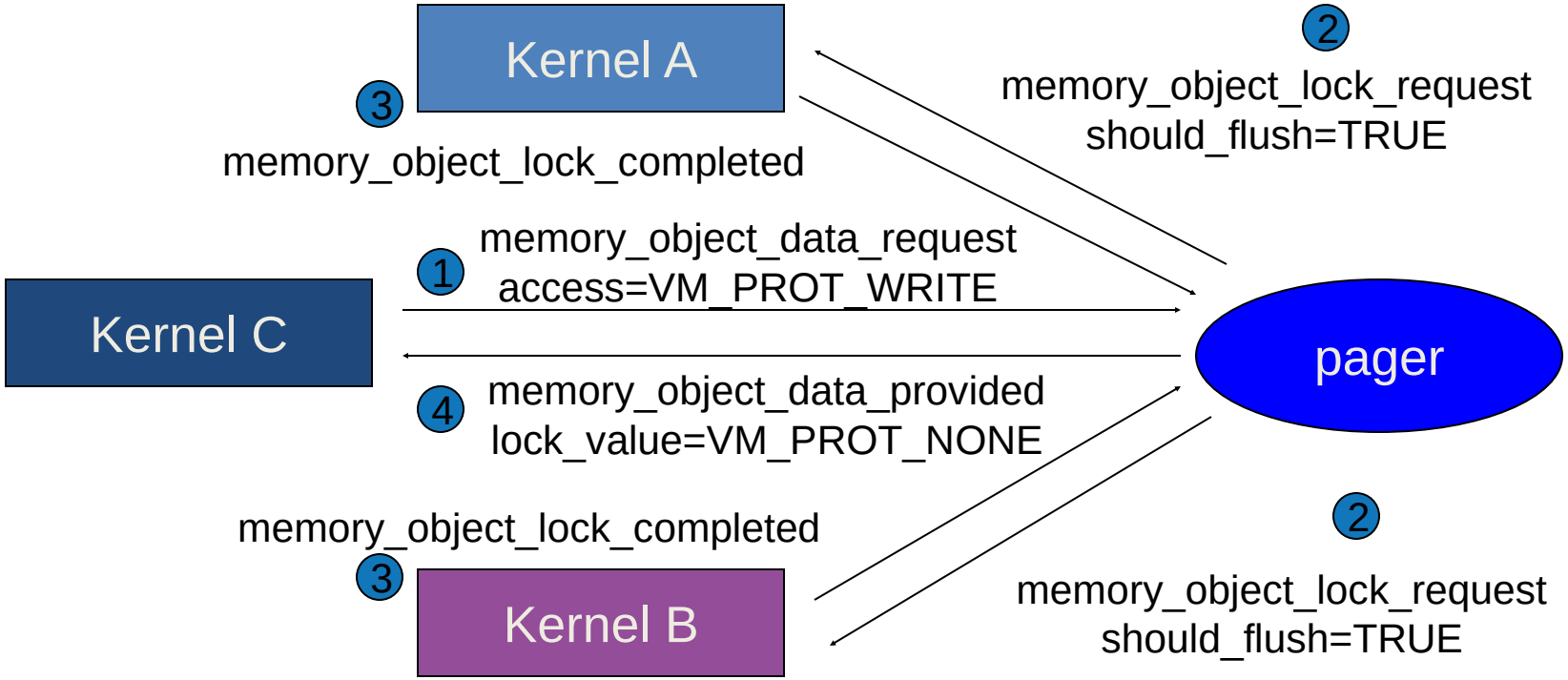
Paging handled by user-space server

Port: comm. endpoint, network-wide



Mach Microkernel OS Distributed Shared Memory System

- Access remote memories, port access rights - ACL



Windows NT Origins

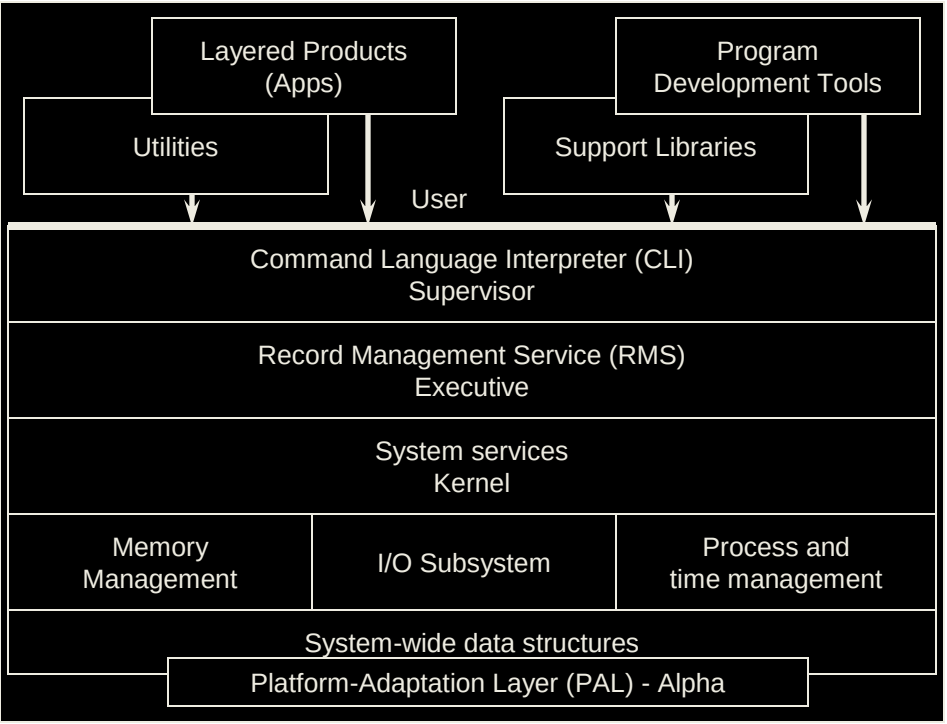
- **Design began in late 1988/early 1989 after Dave Cutler and a handful of Digital employees started at Microsoft**
  - Dave Cutler—legend in the operating system world
    - Project leader for Digital's VMS (Virtual Memory System)
  - Internally, Windows NT has many similarities to Digital's VMS (scheduling, memory management, I/O and driver model)
  - VMS+1=WNT just a coincidence
- **Original goal was replacement for OS/2**
  - **Later goal changed to be the replacement for Windows 3.0**
- The name "Windows NT" was chosen because
  - NT stands for New Technology
    - But at a high level, the architecture and user interface are not really that "new" (as compared to most 32-bit OS's)
  - Also, the i860 Risc CPU NT was originally targeted at was code named N-Ten

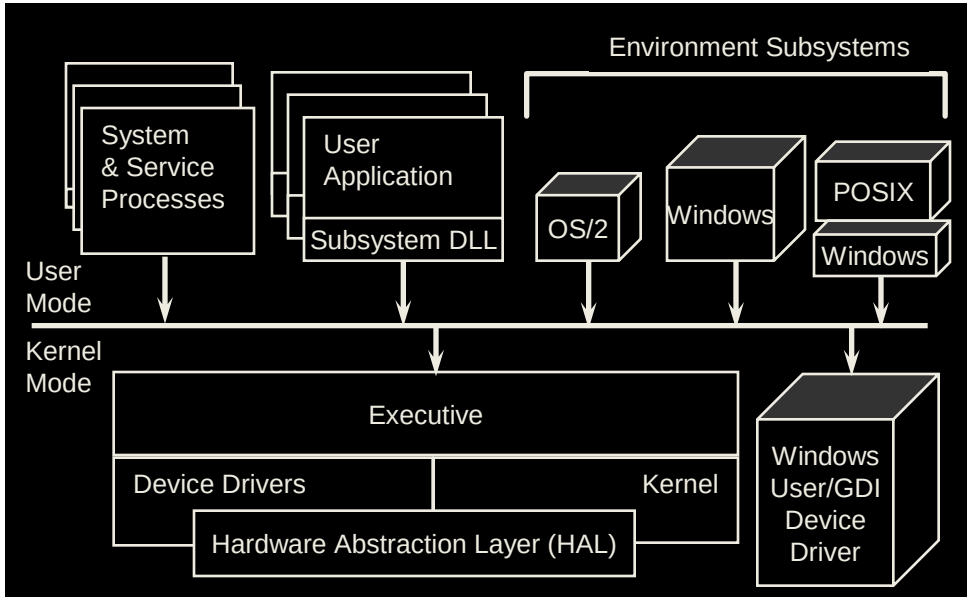
- **Interesting book on the early years of NT:**
  - Show-stopper!: The Breakneck Race to Create Windows NT and the Next Generation at Microsoft
  - By G. Pascal Zachary, ISBN: 0029356717

VMS and Windows

- a bird's-eye view on architectures

Layered design for VAX/VMS operating system





Windows high-level architecture

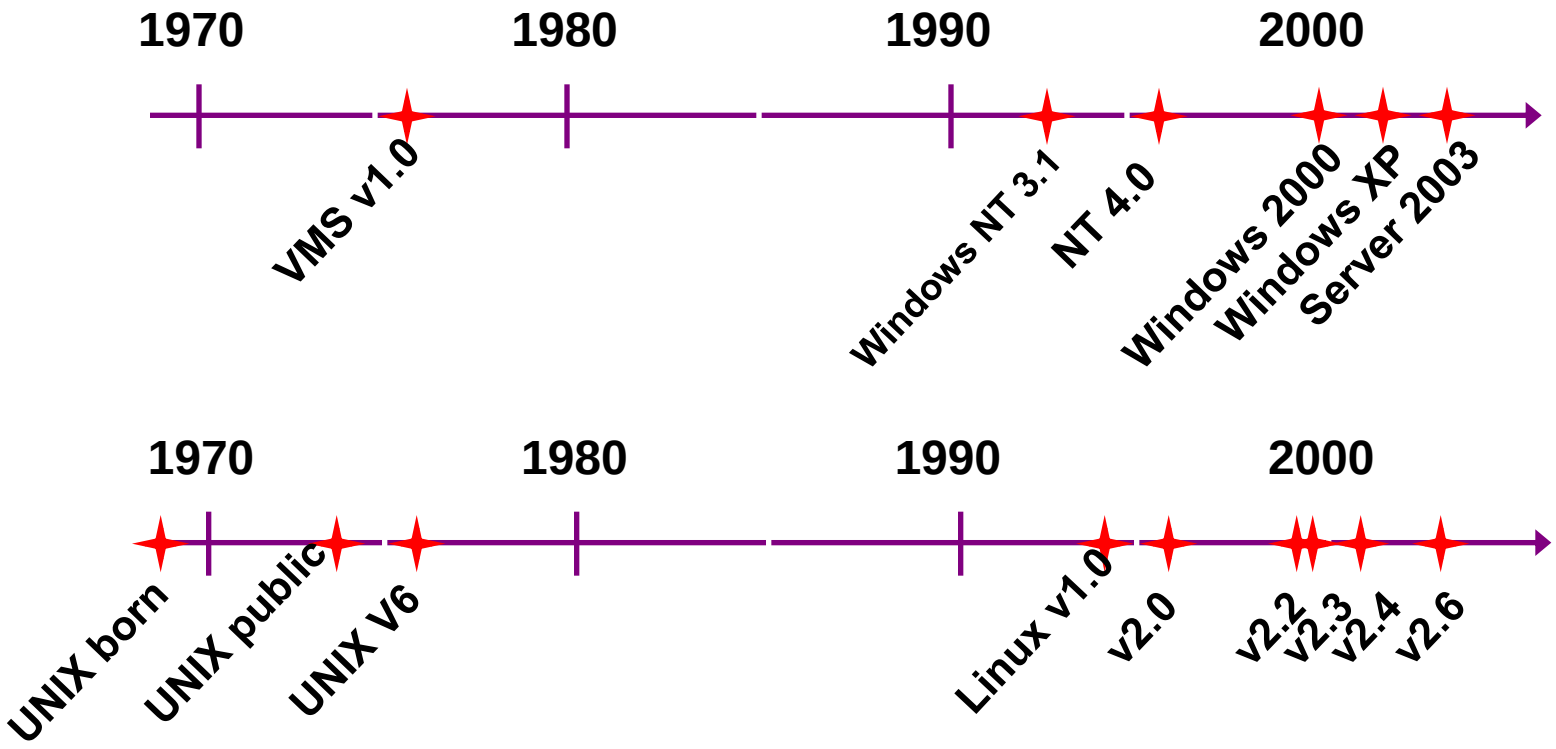
Release History

- Although product name has varied, internally, each version identified by a "build number"
  - Internal identification - increments each time NT is built from source (5-6 times a week)
  - Interesting timeline: http://windows2000.about.com/library/weekly/aa010218a.htm

Build#	Version	Date
297	PDC developer release	Jul 1992
511	NT 3.1	Jul 1993
807	NT 3.5	Sep 1994
1057	NT 3.51	May 1995
1381	NT 4.0	Jul 1996
2195	Windows 2000 (NT 5.0)	Dec 1999
2600	Windows XP (NT 5.1)	Aug 2001
3790	Windows Server 2003 (NT 5.2)	Mar 2003
4051	Longhorn PDC Developer Preview	Oct 2003

Windows And Linux Evolution

- Windows and Linux kernels are based on foundations developed in the mid-1970s



(see http://www.levenez.com for diagrams showing history of Windows & Unix)

Further Reading

- Dennis M. Ritchie, The Evolution of the Unix Time-sharing System,
  - in Proc. of Lang. Design and Programming Meth. Conf., Sydney, Australia, Sept 1979, Lecture Notes in Computer Science #79, Springer-Verlag, 1980.
- David Donald Miller, OpenVMS Operating System Concepts,
  - 2nd Ed., Digital Press, 1997.
  - History of Digital Operating Systems (from pp. 447)
- Mark E. Russinovich and David A. Solomon, Microsoft Windows Internals,
  - 4th Edition, Microsoft Press, 2004.
  - Historical Perspective (from pp. xix)
- G. Pascal Zachary, Show Stopper! The Breakneck Race to Create Windows NT and the Next Generation at Microsoft,
  - ISBN: 0029356717, Free Press, 1994.

Computer-System Structures

- Operating System Services
- User Operating System Interface
- System Calls
- Types of System Calls
- System Programs
- Operating System Design and Implementation
- Operating System Structure
- Virtual Machines
- Operating System Generation
- System Boot

Objectives

- To describe the services an operating system provides to users, processes, and other systems
- To discuss the various ways of structuring an operating system
- To explain how operating systems are installed and customized and how they boot

Operating System Services

- 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

Operating System Services (Cont.)

- 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,mainmemory, 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

- Two principle forms:
  - CLI allows direct command entry
    - Sometimes implemented in kernel, sometimes by systems program
    - Sometimes multiple flavors implemented – **shells**
    - 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)

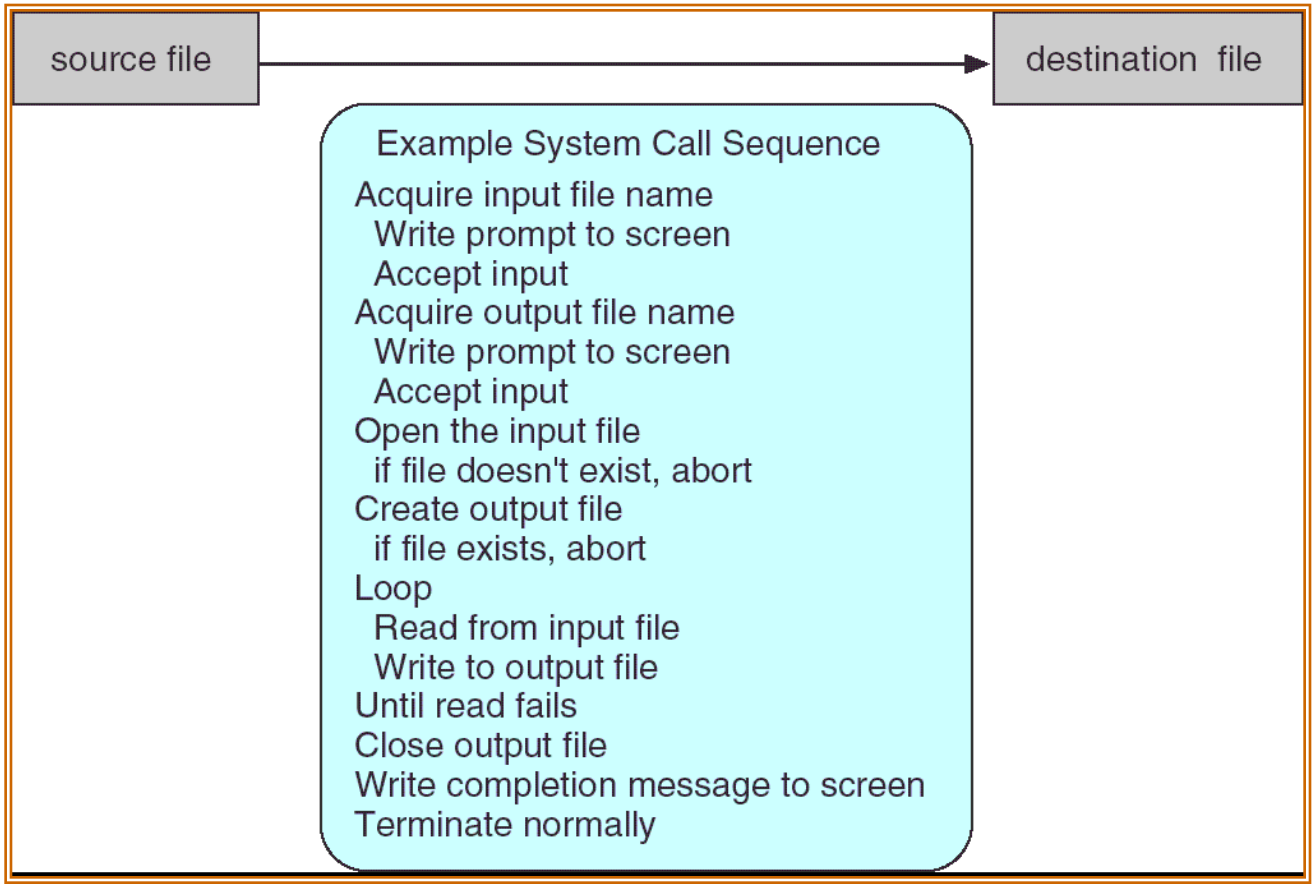
System Calls

- Programming interface to the services provided by the OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level **Application Program Interface (API)** rather than direct system call use
- Three most common APIs are Win32 API for Windows, POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM)
- Why use APIs rather than system calls?

(Note that the system-call names used are generic)

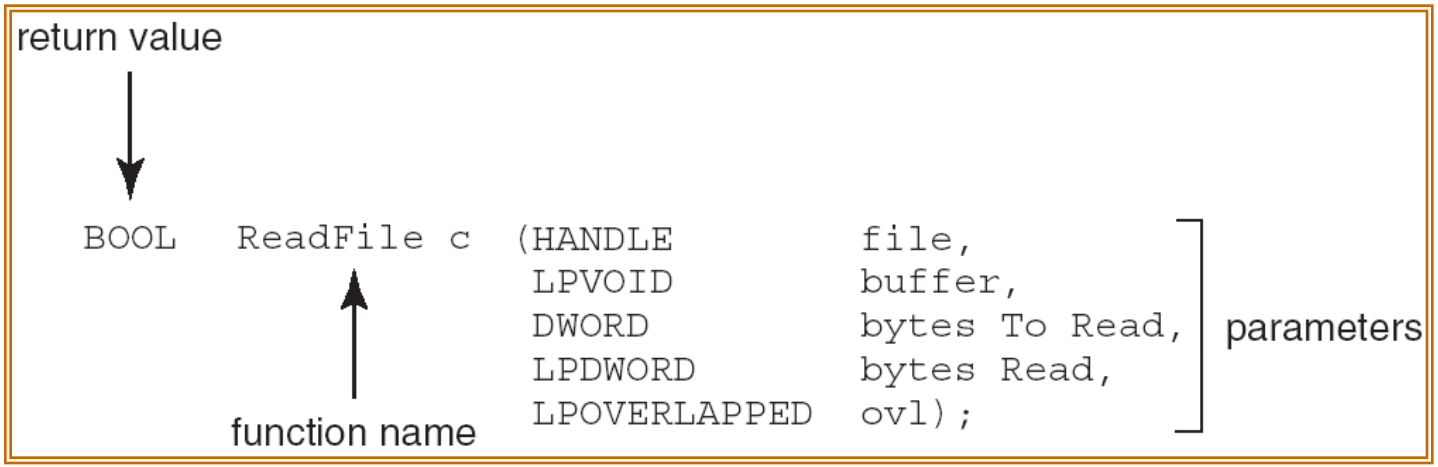
Example of System Calls

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



Example of Standard API

- Consider the ReadFile() function in the
- Win32 API—a function for reading from a file



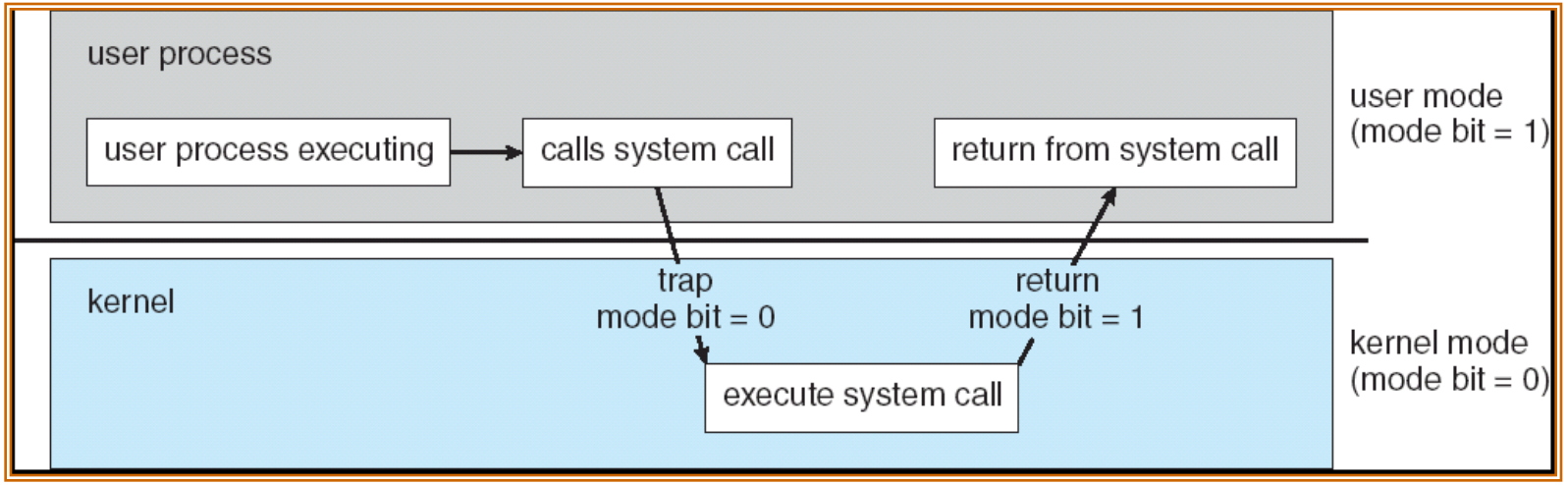
- A description of the parameters passed to ReadFile()
  - HANDLE file—the file to be read
  - LPVOID buffer—a buffer where the data will be read into and written from
  - DWORD bytesToRead—the number of bytes to be read into the buffer
  - LPDWORD bytesRead—the number of bytes read during the last read
  - LPOVERLAPPED ovl—indicates if overlapped I/O is being used

System Call Implementation

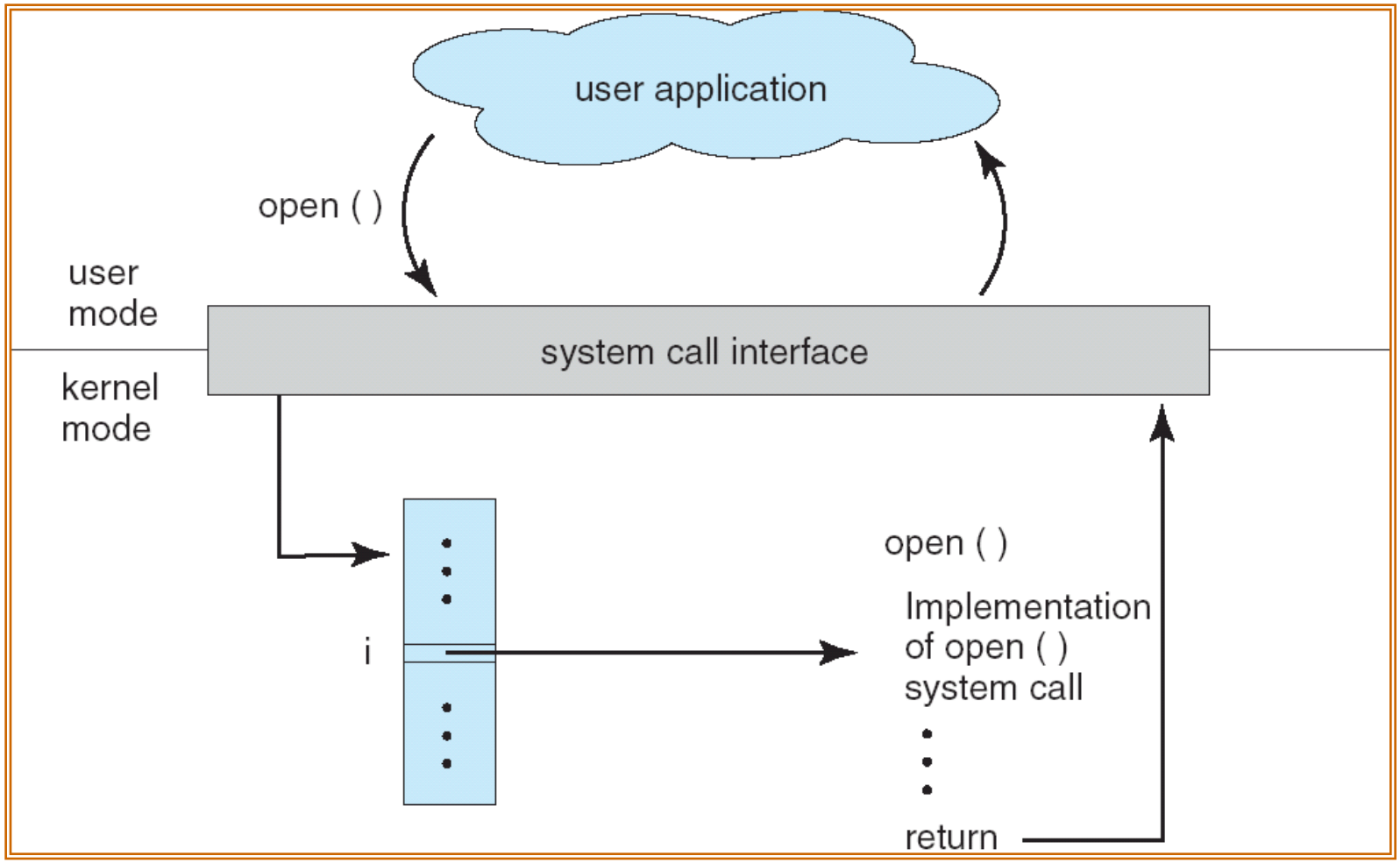
- Typically, a number associated with each system call
  - System-call interface maintains a table indexed according to these numbers
- The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values
- The caller need know nothing about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

Transition from User to Kernel Mode

- Timer to prevent infinite loop / process hogging resources
  - Set interrupt after specific period
  - Operating system decrements counter
  - When counter zero
  - Set up before scheduling process to regain control or terminate program that exceeds allotted time

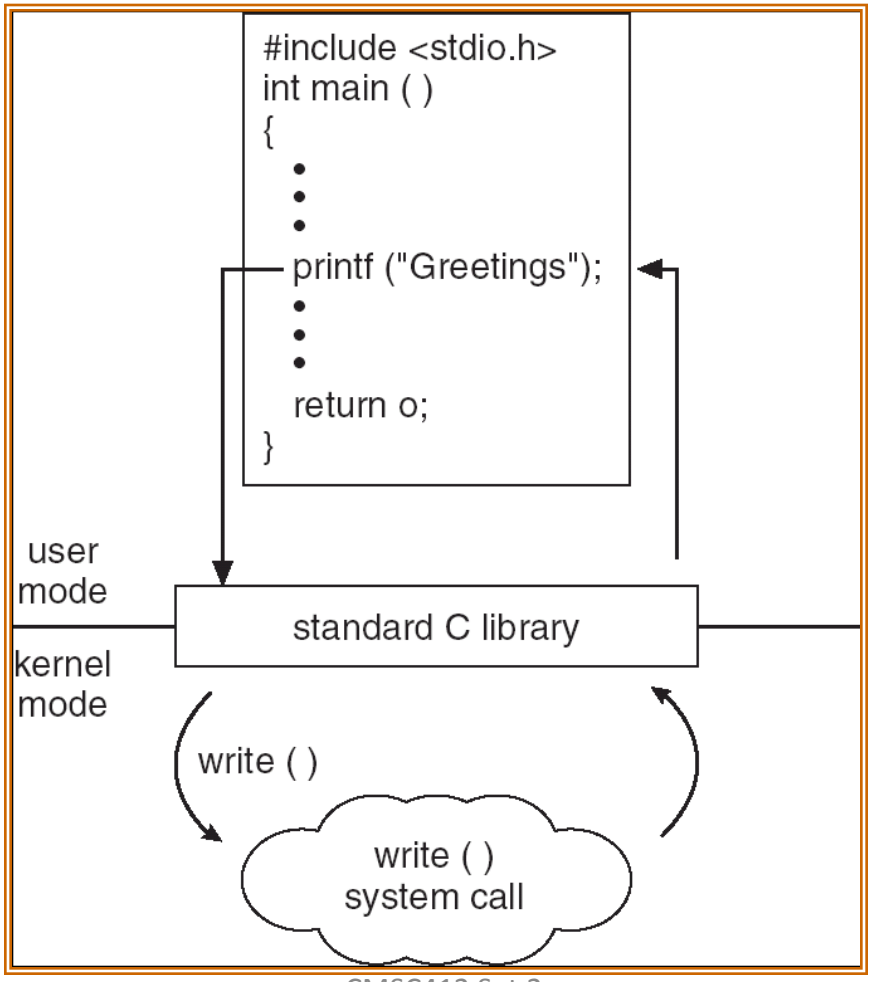


API – System Call – OS Relationship



Standard C Library Example

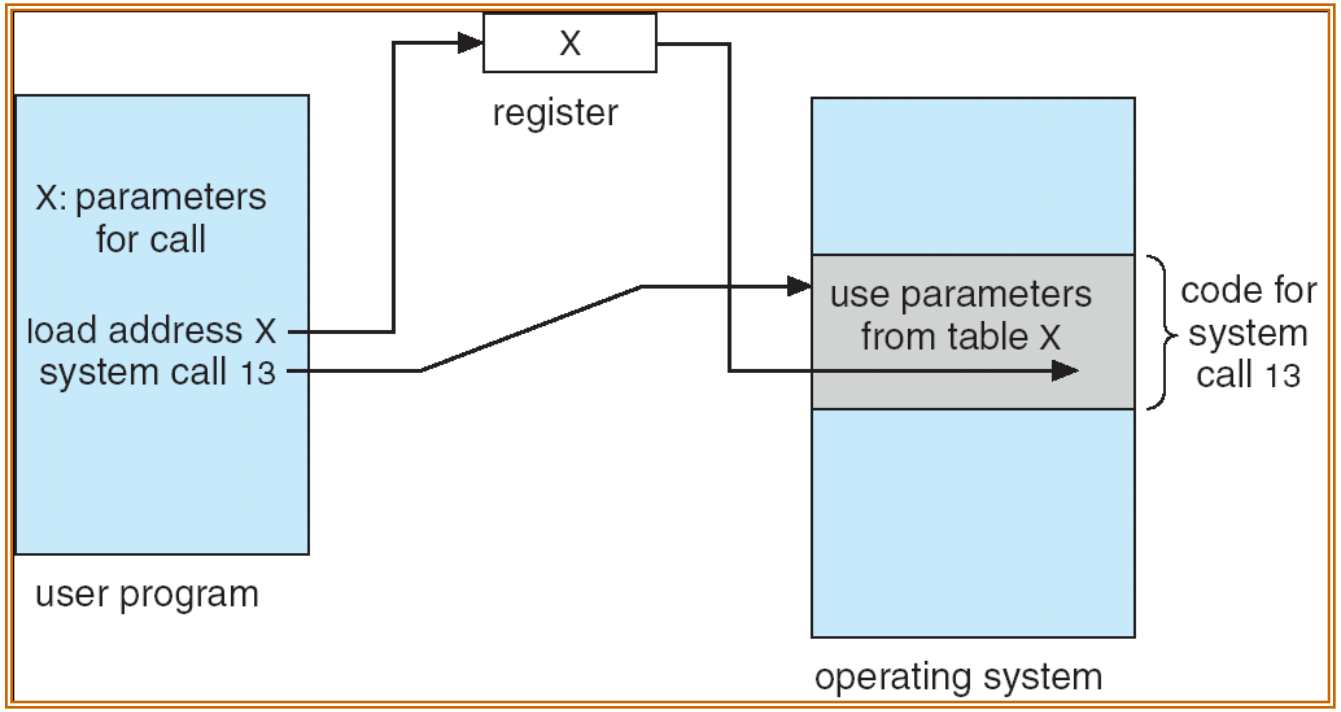
- C program invoking printf() library call, which calls write() system call



System Call Parameter Passing

- Often, more information is required than simply identity of desired system call
  - Exact type and amount of information vary according to OS and call
- Three general methods used to pass parameters to the OS
  - Simplest: pass the parameters in *registers*
    - In some cases, may be more parameters than registers
  - Parameters stored in a *block,* or table, in memory, and address of block passed as a parameter in a register
    - This approach taken by Linux and Solaris
  - Parameters placed, or *pushed,* onto the *stack* by the program and *popped* off the stack by the operating system
  - Block and stack methods do not limit the number or length of parameters being passed

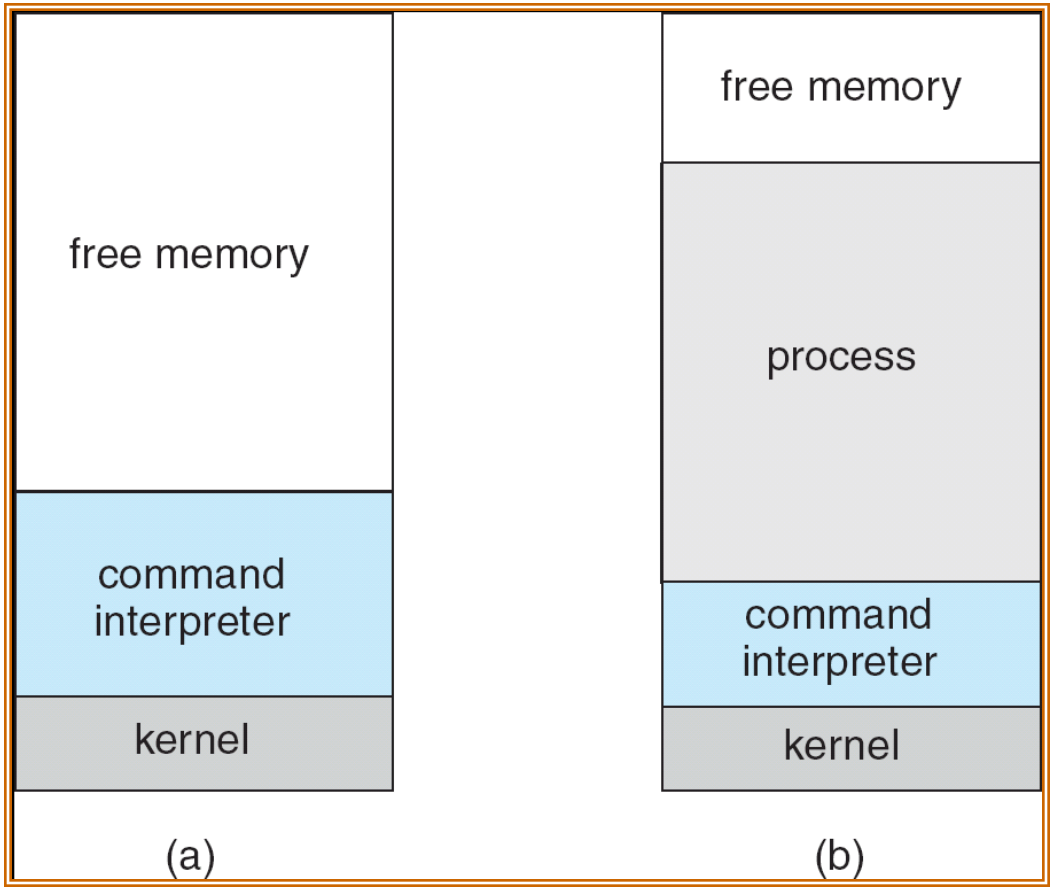
Parameter Passing via Table



Types of System Calls

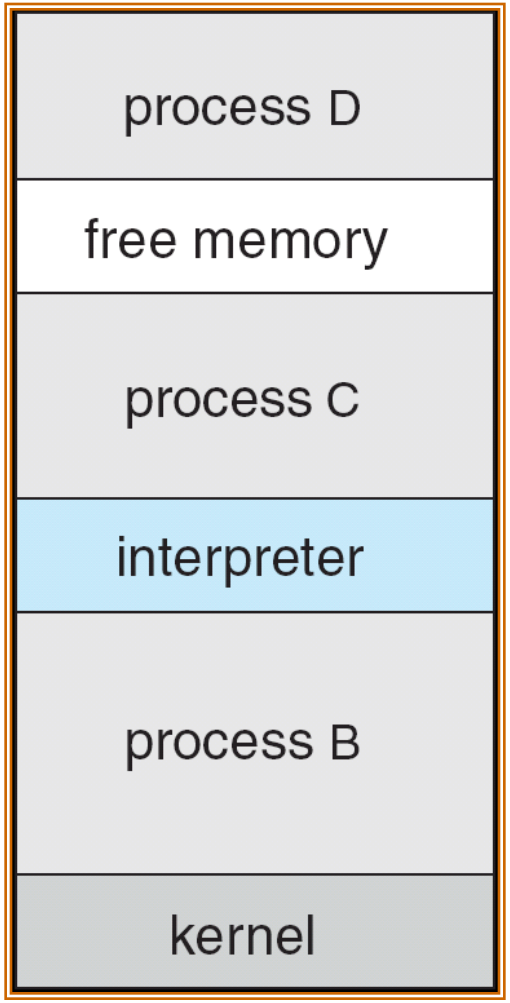
- Process control
- File management
- Device management
- Information maintenance
- Communications

MS-DOS execution



(a) At system startup (b) running a program

FreeBSD Running Multiple Programs



System Programs

- System programs provide a convenient environment for program development and execution. The 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, not the actual system calls

Solaris 10 dtrace Following System Call

```
# ./all.d `pgrep xclock` XEventsQueued
dtrace: script './all.d' matched 52377 probes
CPU FUNCTION
  0 -> XEventsQueued                       U
  0   -> _XEventsQueued                    U
  0     -> _X11TransBytesReadable          U
  0     <- _X11TransBytesReadable          U
  0     -> _X11TransSocketBytesReadable    U
  0     <- _X11TransSocketBytesreadable    U
  0     -> ioctl                           U
  0       -> ioctl                         K
  0         -> getf                        K
  0           -> set_active_fd             K
  0           <- set_active_fd             K
  0         <- getf                        K
  0         -> get_udatamodel              K
  0         <- get_udatamodel              K
...
  0         -> releasef                    K
  0           -> clear_active_fd           K
  0           <- clear_active_fd           K
  0           -> cv_broadcast              K
  0           <- cv_broadcast              K
  0         <- releasef                    K
  0       <- ioctl                         K
  0     <- ioctl                           U
  0   <- _XEventsQueued                    U
  0 <- XEventsQueued                       U
```

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

System Programs (cont'd)

- 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- Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language
- 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

Operating System Design and Implementation

- Design and Implementation of OS not “solvable”, but some approaches have proven successful
- 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
- Important principle to separate

  Policy: What will be done?
  Mechanism: How to do it?
- Mechanisms determine how to do something, policies decide what will be done
  - The separation of policy from mechanism is a very important principle, it allows maximum flexibility if policy decisions are to be changed later

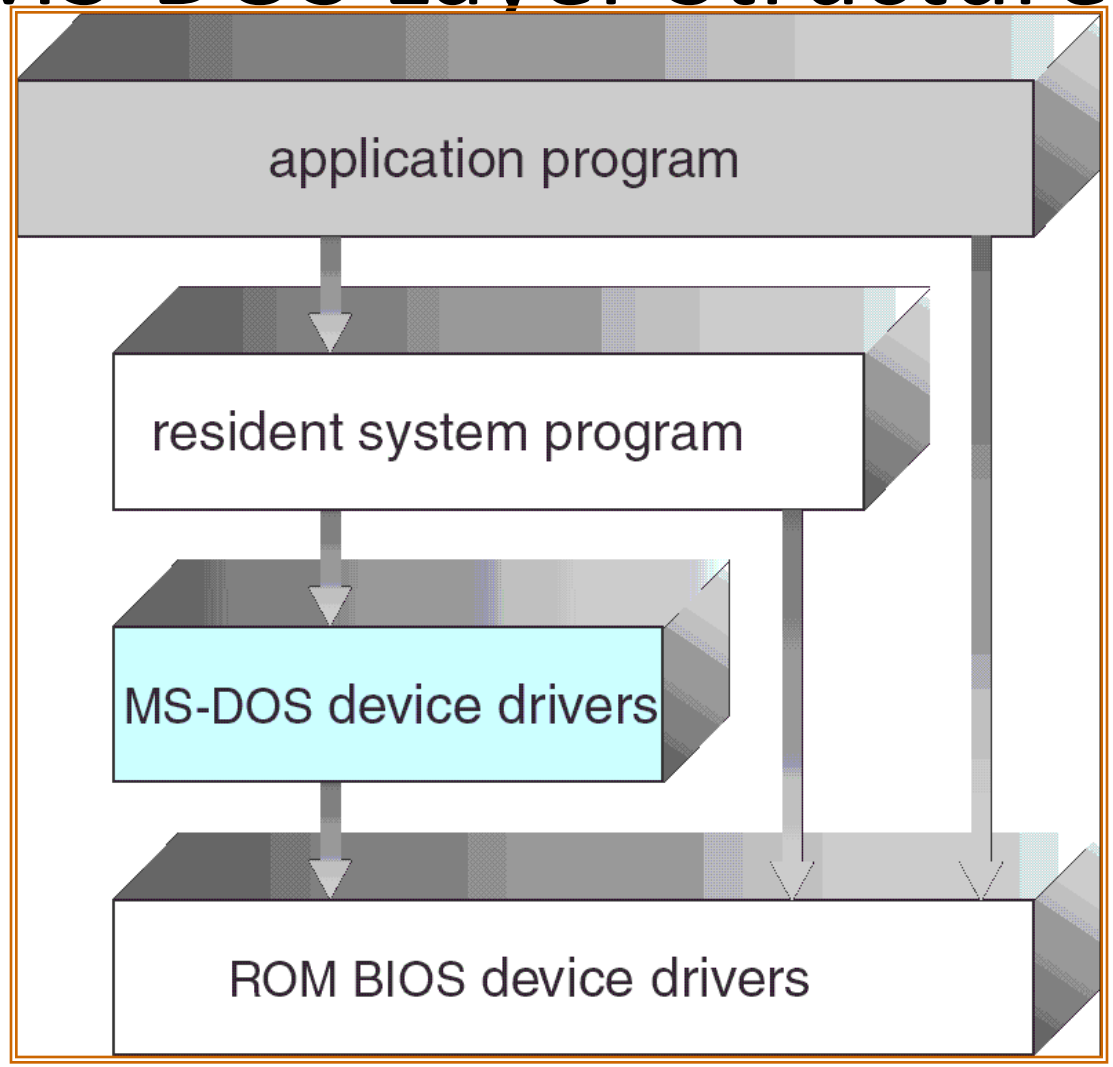
Operating System Structure

- Internal structure of different OSes can vary widely

Simple Structure

- MS-DOS – written to provide the most functionality in the least space
  - Not divided into modules
  - Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated

MS-DOS Layer Structure



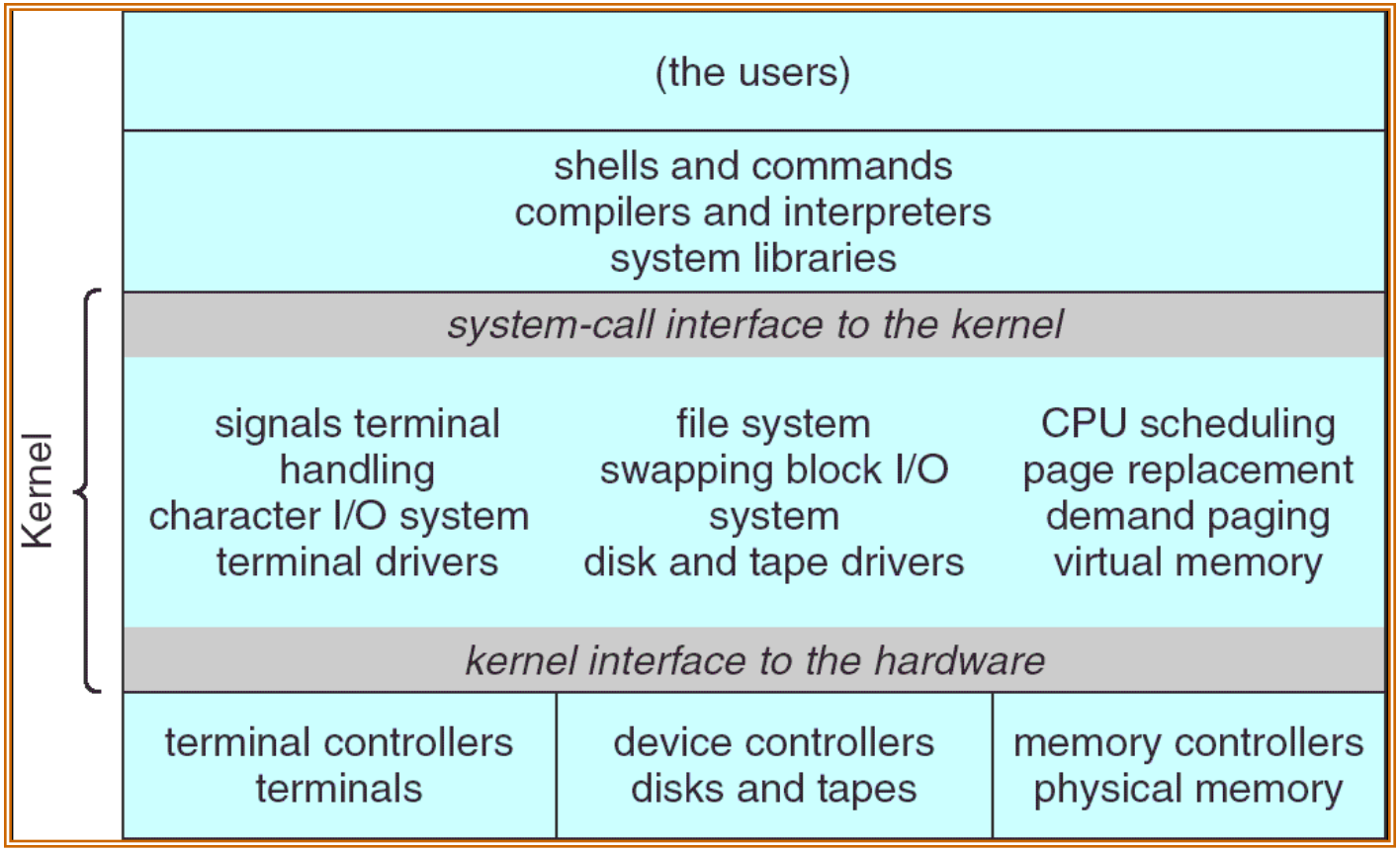
Layered Approach

- The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers

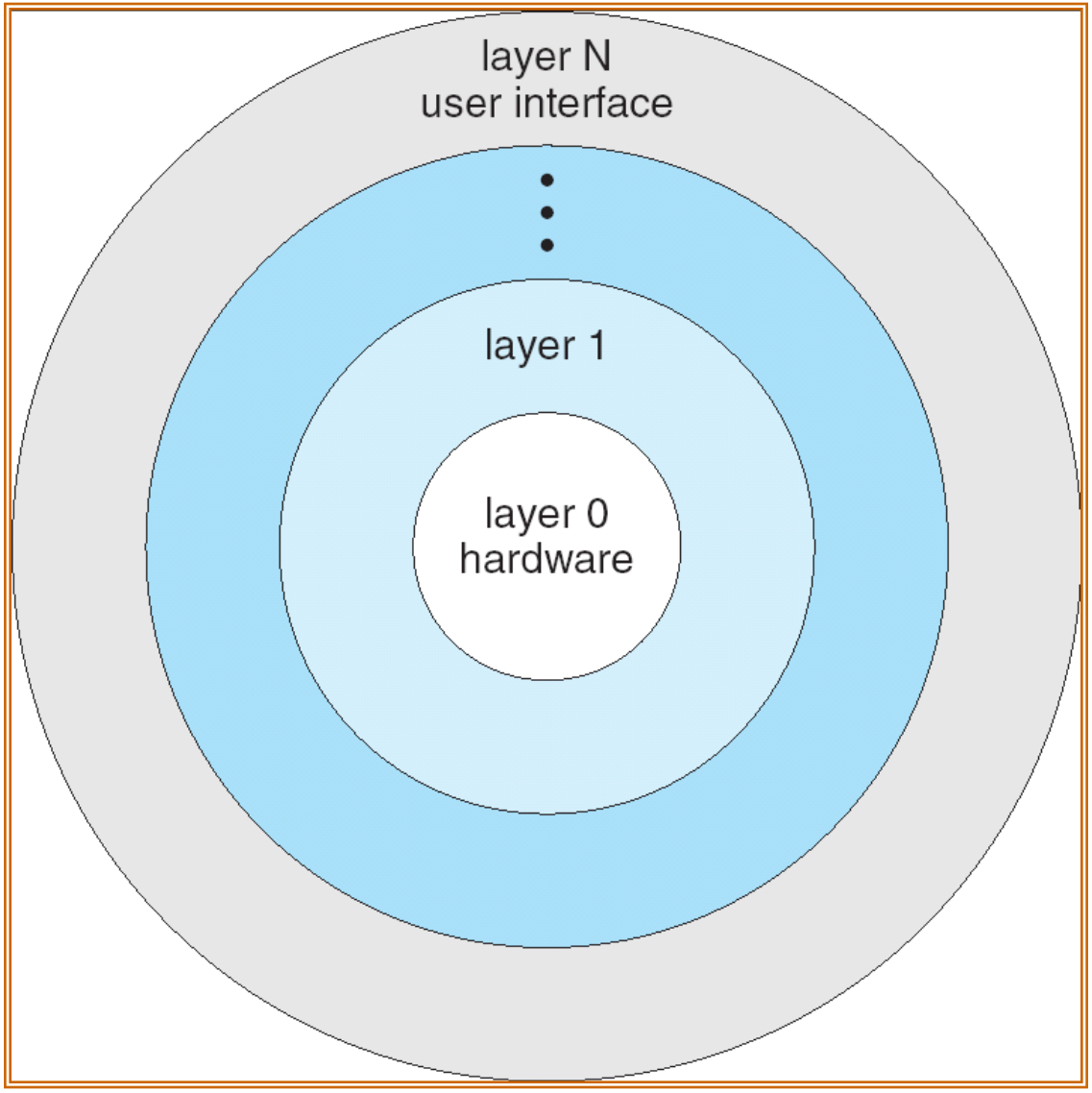
UNIX

- UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring. The UNIX OS consists of two separable parts
  - Systems programs
  - The kernel
    - Consists of everything below the system-call interface and above the physical hardware
    - Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level

UNIX System Structure



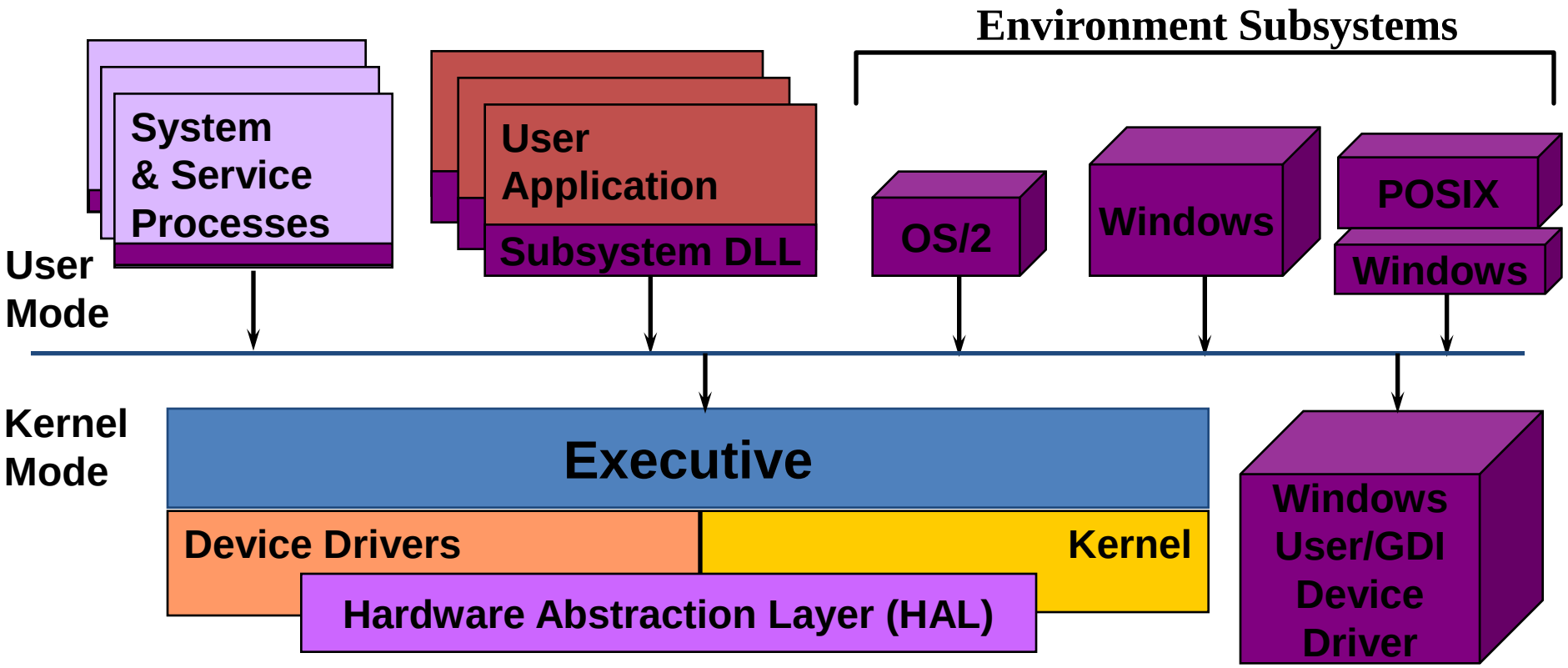
Layered Operating System



Microkernel System Structure

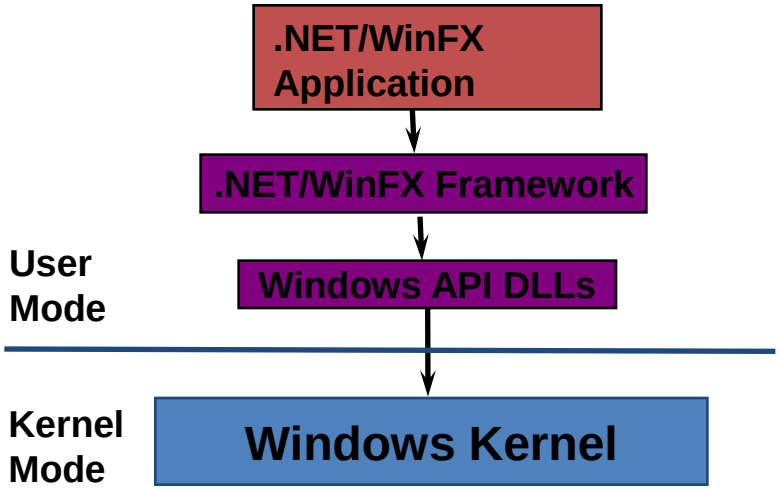
- Moves as much from the kernel into "*user*" space
- Communication takes place between user modules using message passing
- Benefits:
  - Easier to extend a microkernel
  - Easier to port the operating system to new architectures
  - More reliable (less code is running in kernel mode)
  - More secure
- Detriments:
  - Performance overhead of user space to kernel space communication

Multiple OS Personalities

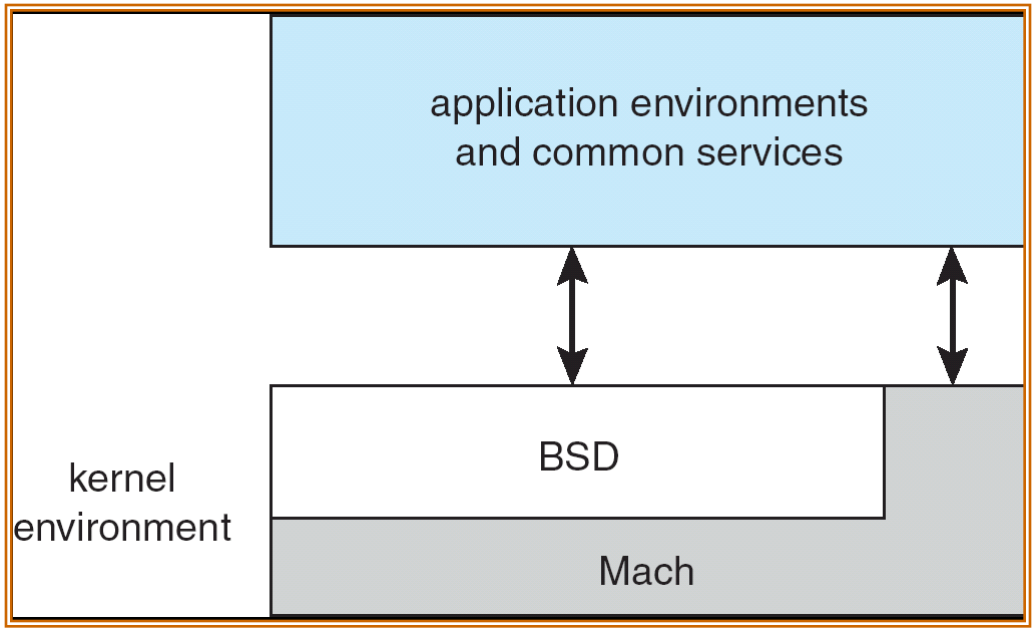


What about .NET and WinFX?

- WinFX is the .NET Framework that will ship with Longhorn
- Both .NET and WinFX are built on standard Windows APIs
  - They are not a subsystem
  - They do not call undocumented Windows system calls



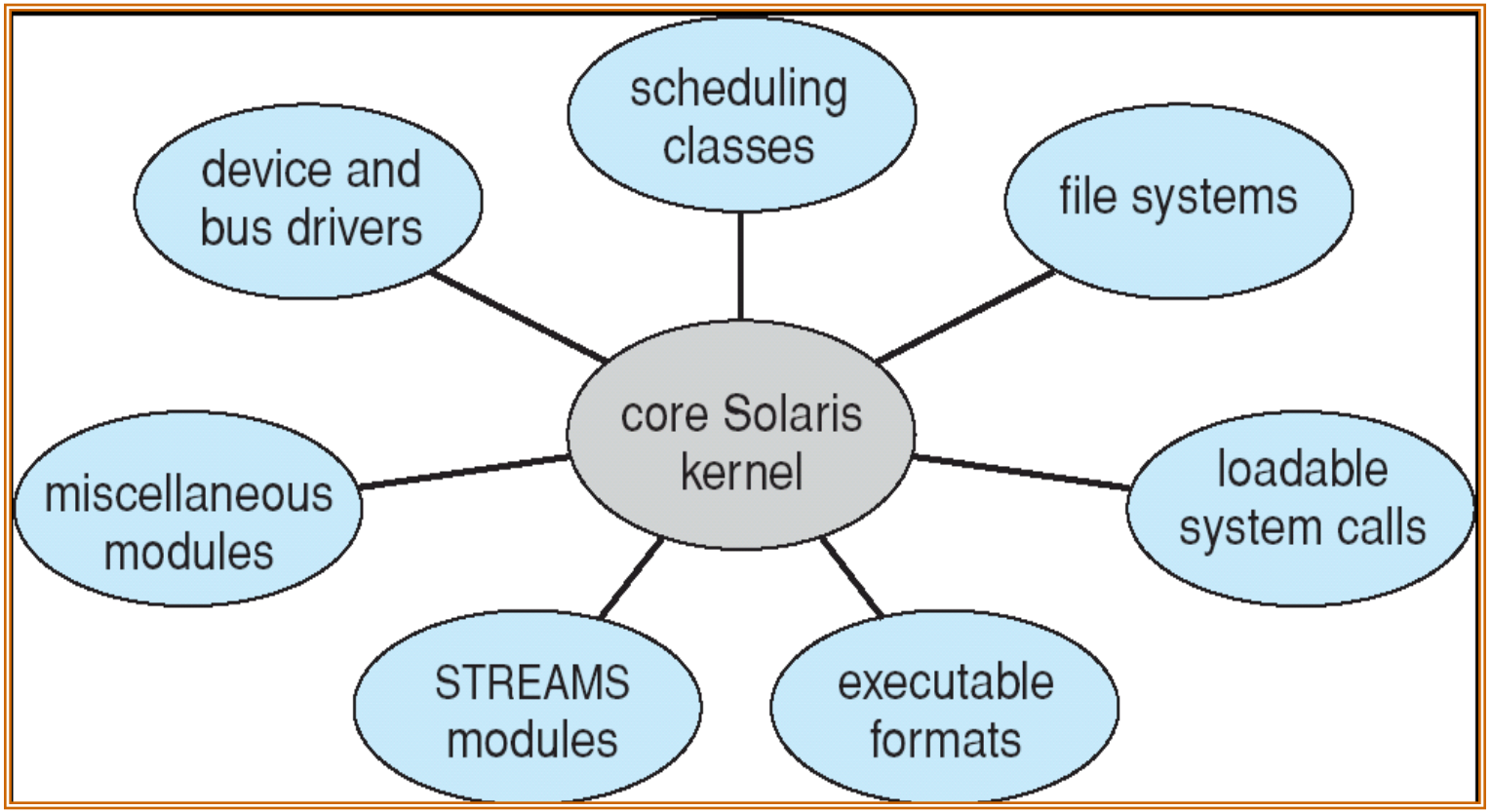
Mac OS X Structure



Modules

- Most modern operating systems implement kernel *modules*
  - Uses object-oriented approach
  - Each core component is separate
  - Each talks to the others over known interfaces
  - Each is loadable as needed within the kernel
- Overall, similar to layers but with more flexible

Solaris Modular Approach



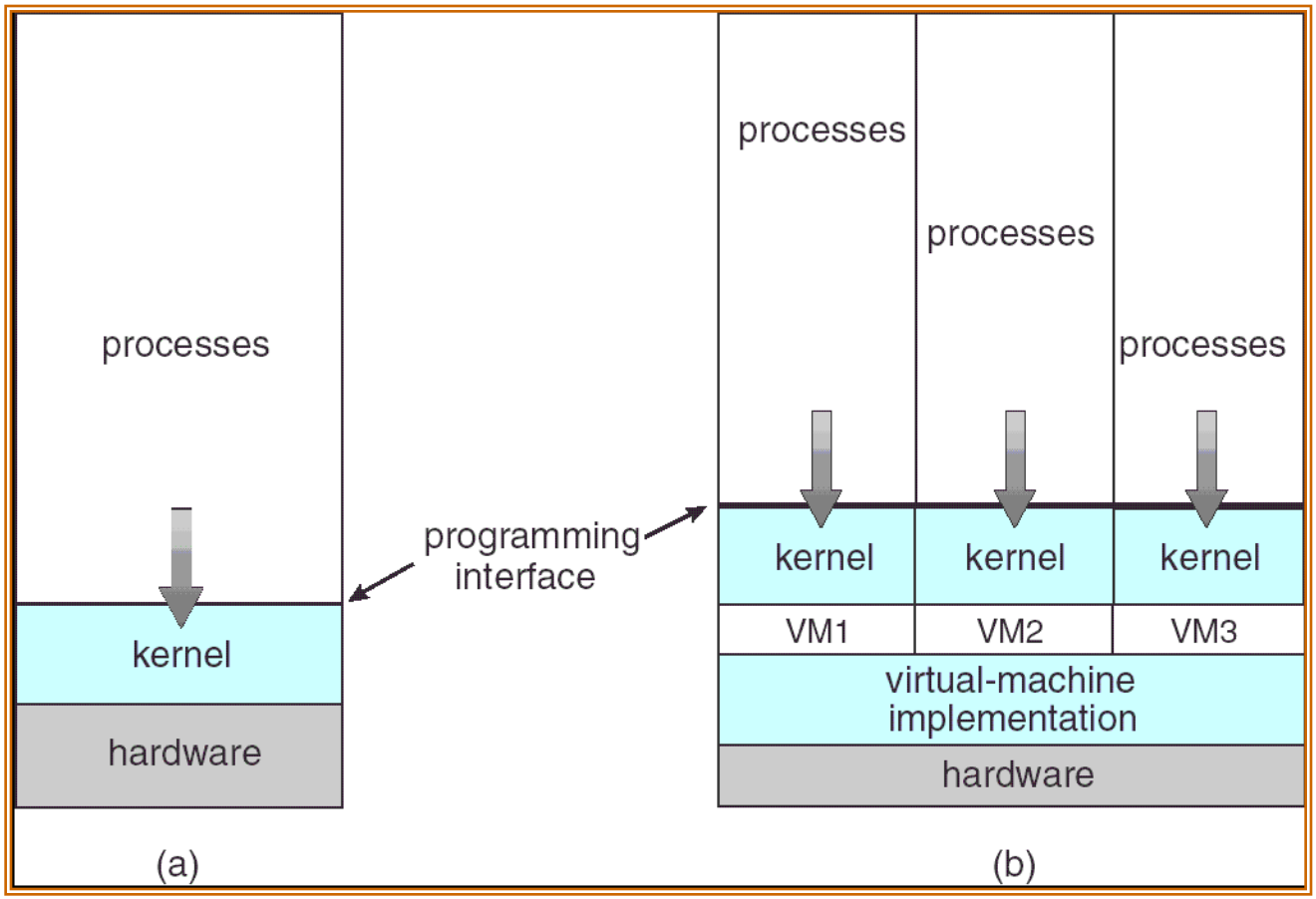
Virtual Machines

- A *virtual machine* takes the layered approach to its logical conclusion. It treats hardware and the operating system kernel as though they were all hardware
- A virtual machine provides an interface *identical* to the underlying bare hardware
- The operating system creates the illusion of multiple processes, each executing on its own processor with its own (virtual) memory

Virtual Machines (Cont.)

- The resources of the physical computer are shared to create the virtual machines
  - CPU scheduling can create the appearance that users have their own processor
  - Spooling and a file system can provide virtual card readers and virtual line printers
  - A normal user time-sharing terminal serves as the virtual machine operator's console

System Models

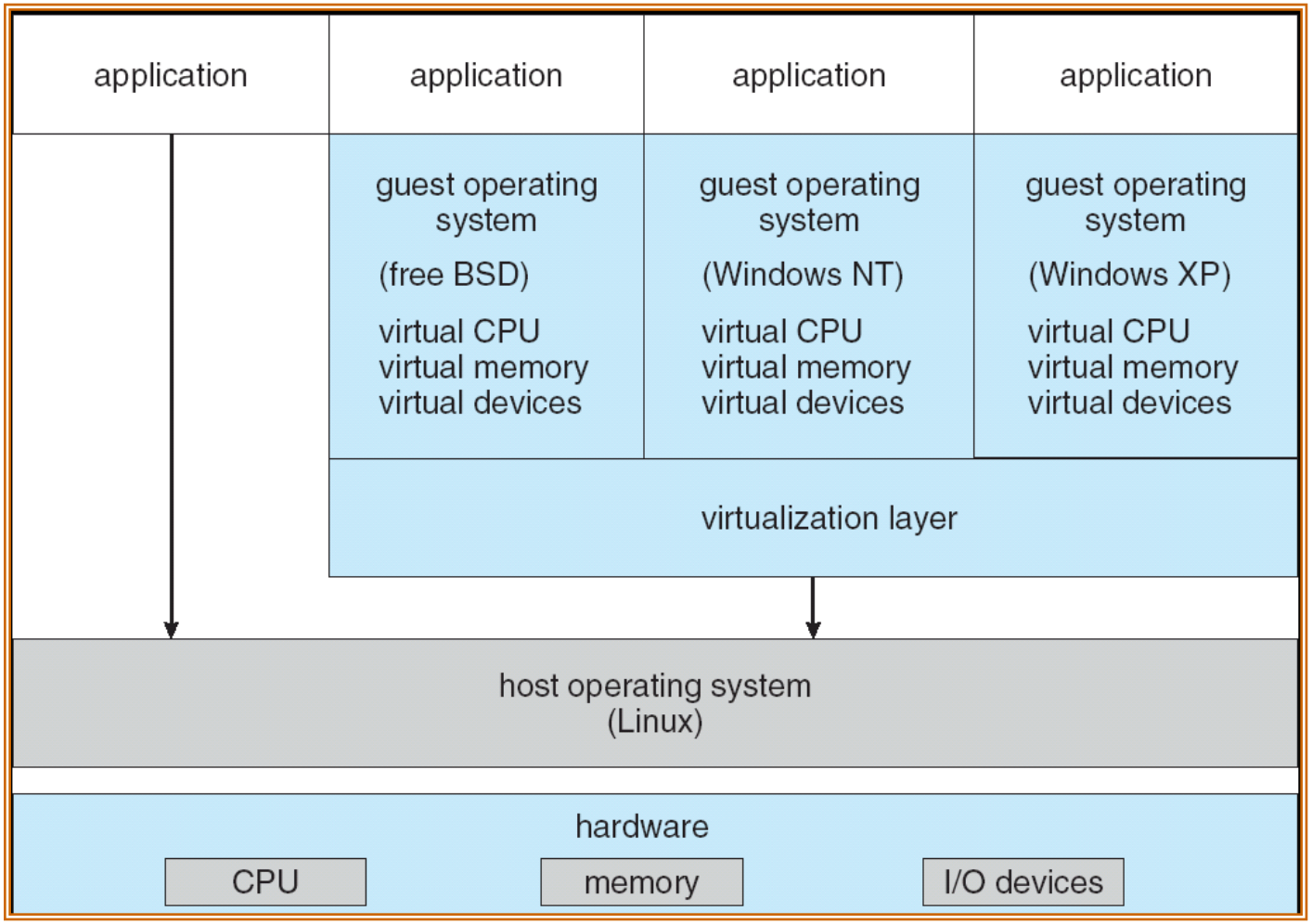


(a) Nonvirtual machine (b) virtual machine

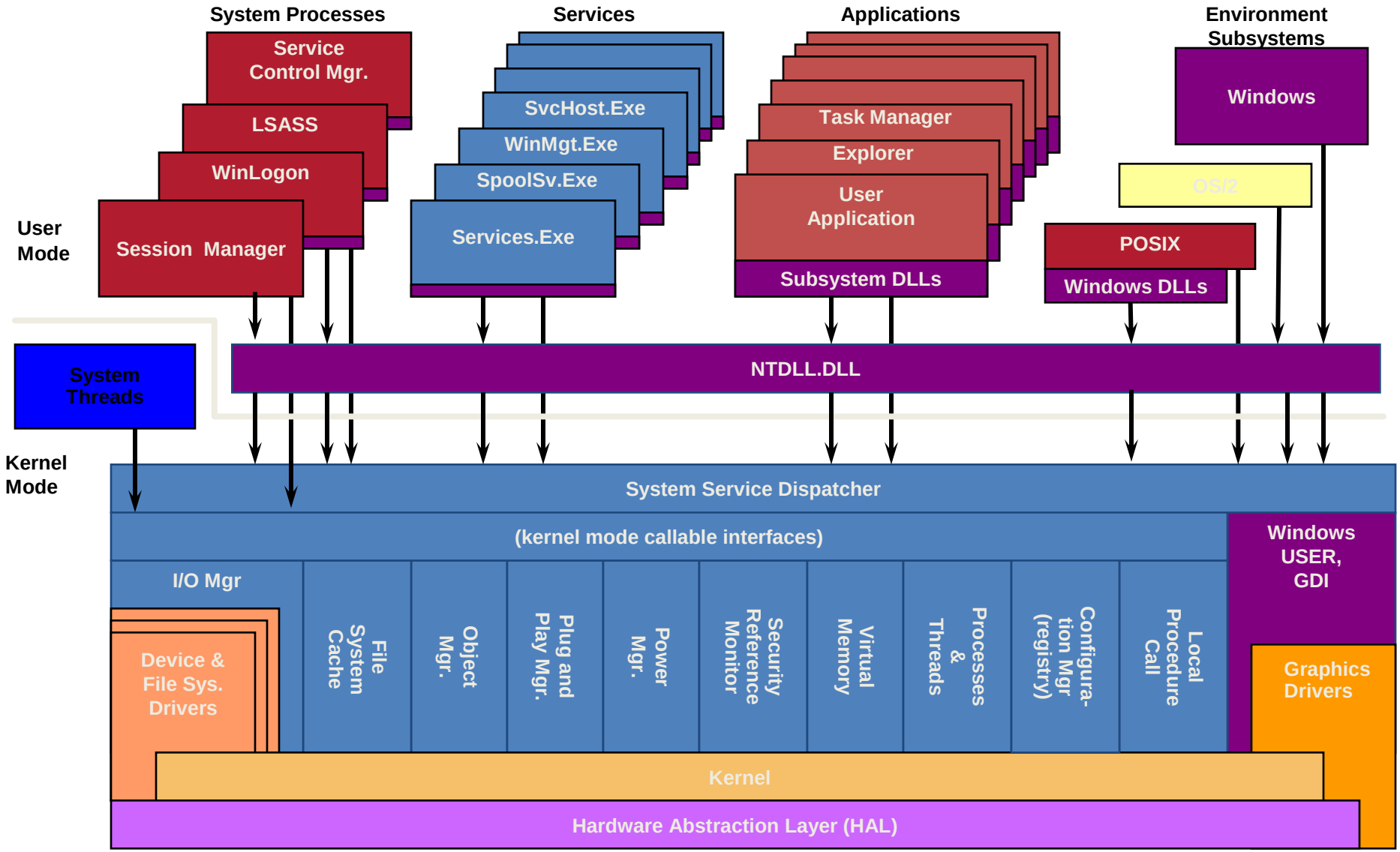
Advantages/Disadvantages of Virtual Machines

- The virtual-machine concept provides complete protection of system resources since each virtual machine is isolated from all other virtual machines. This isolation, however, permits no direct sharing of resources.
- A virtual-machine system is a perfect vehicle for operating-systems research and development. System development is done on the virtual machine, instead of on a physical machine and so does not disrupt normal system operation.
- The virtual machine concept is difficult to implement due to the effort required to provide an *exact* duplicate to the underlying machine

VMware Architecture



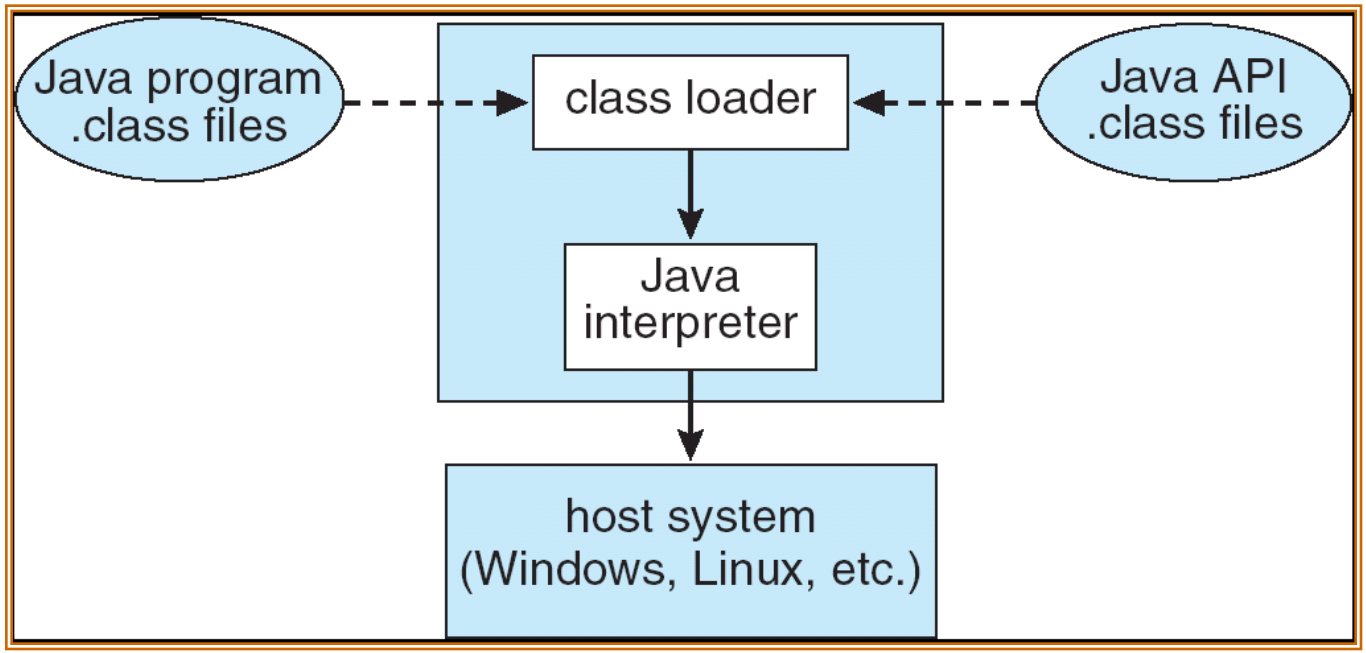
Windows Architecture



hardware interfaces (buses, I/O devices, interrupts, interval timers, DMA, memory cache control, etc., etc.)

Original copyright by Microsoft Corporation. Used by permission.

The Java Virtual Machine



Operating System Generation

- Operating systems are designed to run on any of a class of machines; the system must be configured for each specific computer site
- SYSGEN program obtains information concerning the specific configuration of the hardware system
- *Booting* – starting a computer by loading the kernel
- *Bootstrap program* – code stored in ROM that is able to locate the kernel, load it into memory, and start its execution

System Boot

- Operating system must be made available to hardware so hardware can start it
  - Small piece of code – **bootstrap loader**, locates the kernel, loads it into memory, and starts it
  - Sometimes two-step process where **boot block** at fixed location loads bootstrap loader
  - When power initialized on system, execution starts at a fixed memory location
    - Firmware used to hold initial boot code