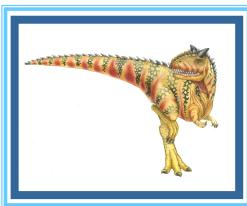
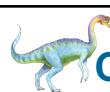


Chapter 2: Operating-System Structures



Operating System Concepts – 8th Edition

Silberschatz, Galvin and Gagne ©2009



Chapter 2: Operating-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 Debugging
- Operating System Generation
- System Boot



Operating System Concepts – 8th Edition

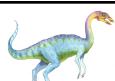
2.2

Silberschatz, Galvin and Gagne ©2009



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

- Operating systems provide an environment for execution of programs and services to programs and users
- 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. Programs need to read and write files and directories, create and delete them, search them, list file information, permission management.





Operating System Services (Cont.)

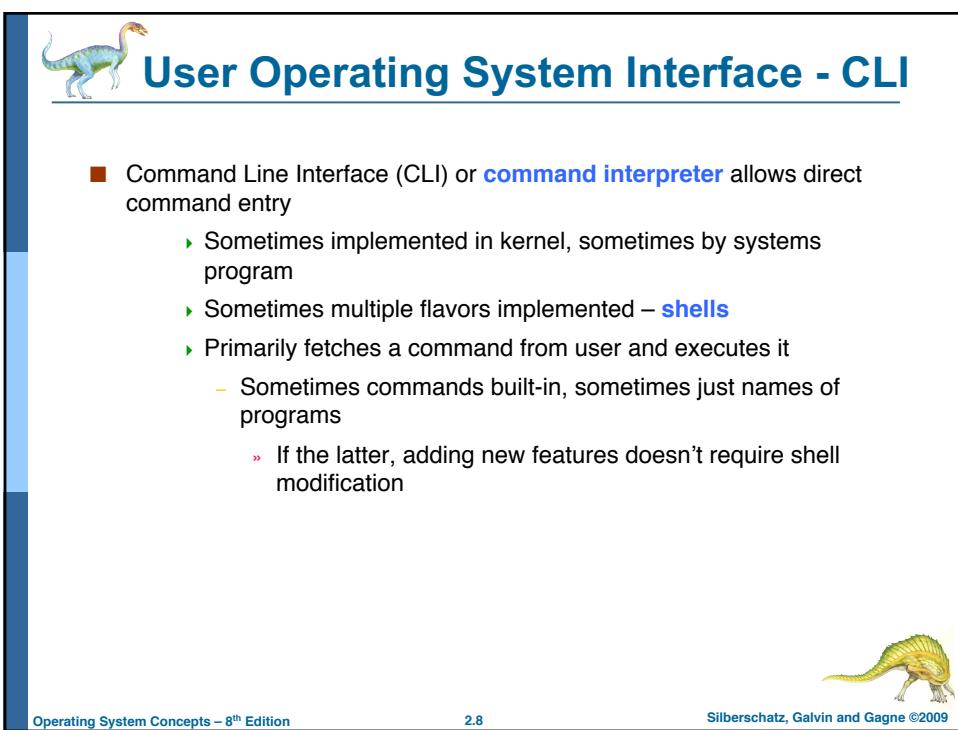
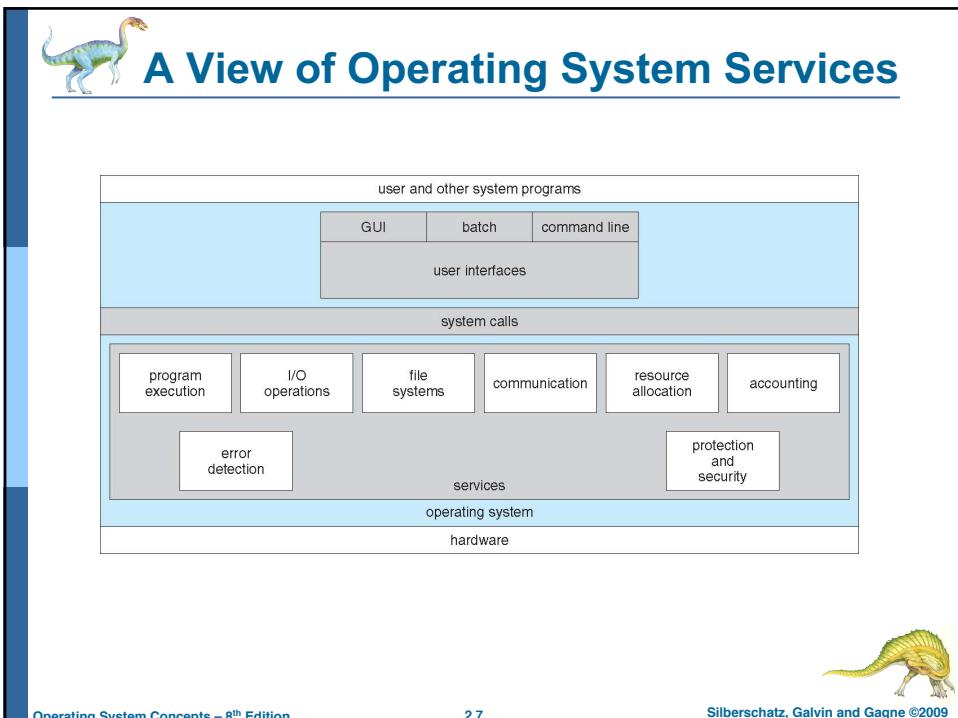
- **Communications** – Processes may exchange information, on the same computer or between computers over a network
 - ▶ Communications may be via shared memory or through message passing (packets moved by the OS)
- **Error detection** – OS needs to be constantly aware of possible errors
 - ▶ May occur 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, 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 - 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

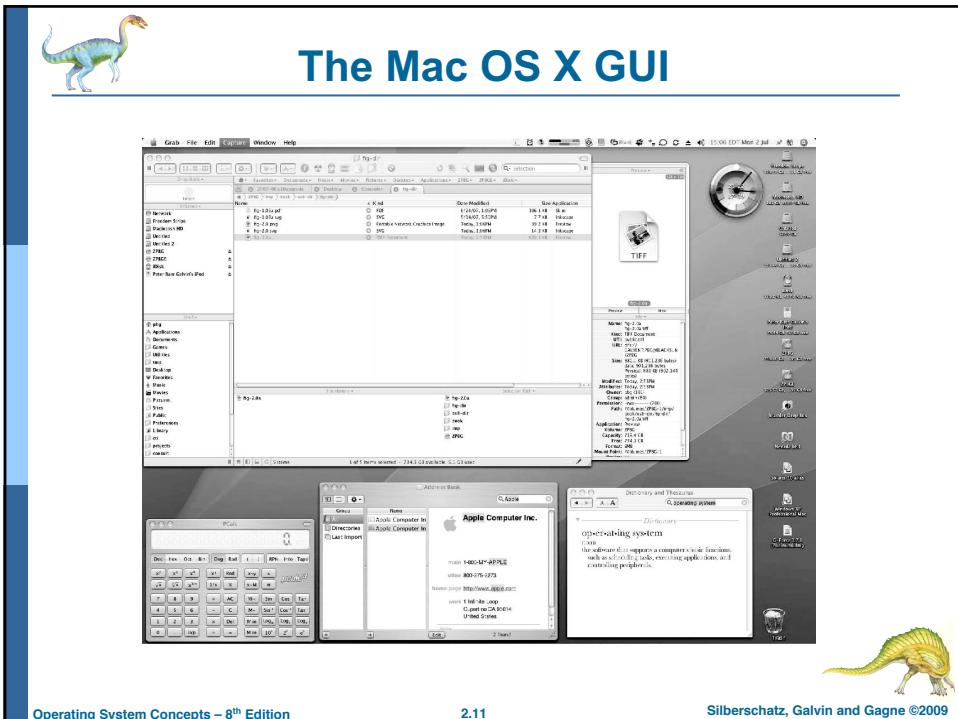
- Many systems now include both CLI and GUI interfaces
 - Microsoft Windows is GUI with CLI "command" shell
 - Apple Mac OS X as "Aqua" GUI interface with UNIX kernel underneath and shells available
 - Solaris is CLI with optional GUI interfaces (Java Desktop, KDE)



Bourne Shell Command Interpreter

```
Terminal
File Edit View Terminal Tabs Help
fd0      0.0    0.0    0.0    0.0    0.0    0.0    0.0    0  0
sd0      0.0    0.2    0.0    0.2    0.0    0.0    0.4    0  0
sd1      0.0    0.0    0.0    0.0    0.0    0.0    0.0    0  0
          extended device statistics
device   r/s    w/s    kr/s   kw/s  wait  activ  svc_t %w  %b
fd0      0.0    0.0    0.0    0.0    0.0    0.0    0.0    0  0
sd0      0.6    0.0    38.4   0.0    0.0    0.0    8.2    0  0
sd1      0.0    0.0    0.0    0.0    0.0    0.0    0.0    0  0
(root@pbg-nv64-vm)-(11/pts)-(00:53 15-Jun-2007)-(global)
-/var/tmp/system-contents/scripts)# swap -sh
total: 1.1G allocated + 190M reserved = 1.3G used, 1.6G available
(root@pbg-nv64-vm)-(12/pts)-(00:53 15-Jun-2007)-(global)
-/var/tmp/system-contents/scripts)# uptime
12:53am up 9 min(s), 3 users, load average: 33.29, 67.68, 36.81
(root@pbg-nv64-vm)-(13/pts)-(00:53 15-Jun-2007)-(global)
-/var/tmp/system-contents/scripts)# w
4:07pm up 17 day(s), 15:24, 3 users, load average: 0.09, 0.11, 8.66
User     tty      login@  idle   JCPU   PCPU what
root    console   15Jun0718days    1    /usr/bin/ssh-agent -- /usr/bi
n/d
root    pts/3    15Jun07           18      4  w
root    pts/4    15Jun0718days           w
(root@pbg-nv64-vm)-(14/pts)-(16:07 02-Jul-2007)-(global)
-/var/tmp/system-contents/scripts#
```





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?

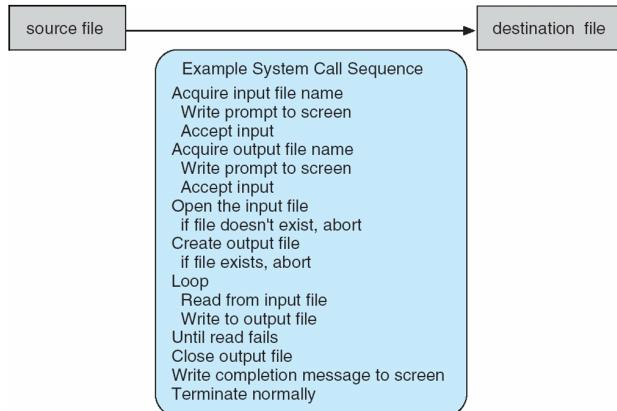
(Note that the system-call names used throughout this text are generic)

Operating System Concepts – 8th Edition 2.12 Silberschatz, Galvin and Gagne ©2009



Example of System Calls

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



Operating System Concepts – 8th Edition

2.13

Silberschatz, Galvin and Gagne ©2009



Example of Standard API

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

```
return value  
↓  
BOOL ReadFile c (HANDLE file,  
                  LPVOID buffer,  
                  DWORD bytes To Read,  
                  DWORD bytes Read,  
                  LPOVERLAPPED ovl);  
↑  
function name
```

- 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



Operating System Concepts – 8th Edition

2.14

Silberschatz, Galvin and Gagne ©2009

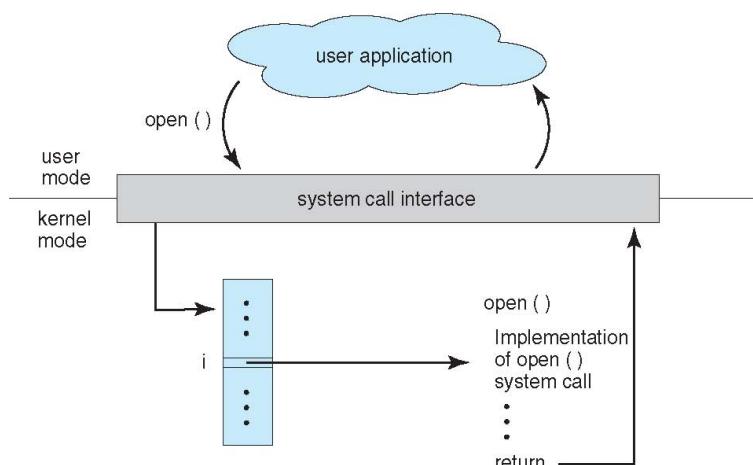


System Call Implementation

- 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)



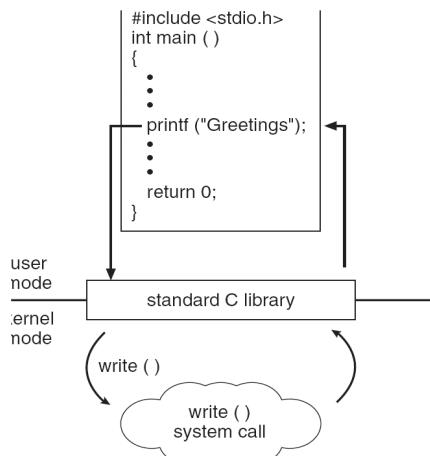
API – System Call – OS Relationship





Standard C Library Example

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



Operating System Concepts – 8th Edition

2.17

Silberschatz, Galvin and Gagne ©2009



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



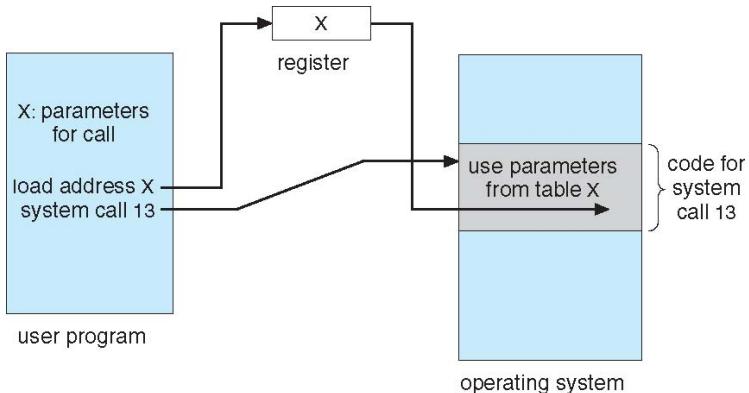
Operating System Concepts – 8th Edition

2.18

Silberschatz, Galvin and Gagne ©2009



Parameter Passing via Table



Operating System Concepts – 8th Edition

2.19

Silberschatz, Galvin and Gagne ©2009



Types of System Calls

- Process control
 - end, abort
 - load, execute
 - create process, terminate process
 - get process attributes, set process attributes
 - wait for time
 - wait event, signal event
 - allocate and free memory
- File management
 - create file, delete file
 - open, close file
 - read, write, reposition
 - get and set file attributes



Operating System Concepts – 8th Edition

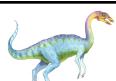
2.20

Silberschatz, Galvin and Gagne ©2009



Types of System Calls (Cont.)

- Device management
 - request device, release device
 - read, write, reposition
 - get device attributes, set device attributes
 - logically attach or detach devices
- Information maintenance
 - get time or date, set time or date
 - get system data, set system data
 - get and set process, file, or device attributes
- Communications
 - create, delete communication connection
 - send, receive messages
 - transfer status information
 - attach and detach remote devices



Examples of Windows and Unix 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() InitializeSecurityDescriptor() SetSecurityDescriptorGroup() | chmod() umask() chown() |



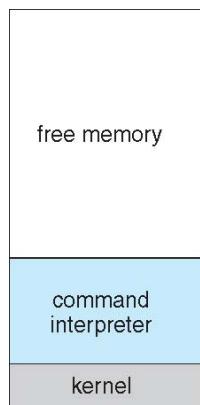


Example: MS-DOS

- Single-tasking
- Shell invoked when system booted
- Simple method to run program
 - No process created
- Single memory space
- Loads program into memory, overwriting all but the kernel
- Program exit -> shell reloaded



MS-DOS execution



(a)



(b)

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



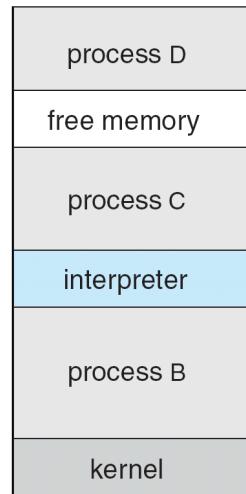


Example: FreeBSD

- Unix variant
- Multitasking
- User login -> invoke user's choice of shell
- Shell executes fork() system call to create process
 - Executes exec() to load program into process
 - Shell waits for process to terminate or continues with user commands
- Process exits with code of 0 – no error or > 0 – error code



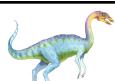
FreeBSD Running Multiple Programs





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, not the actual system calls



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.)

- **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
- 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





Operating System Design and Implementation (Cont.)

- 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



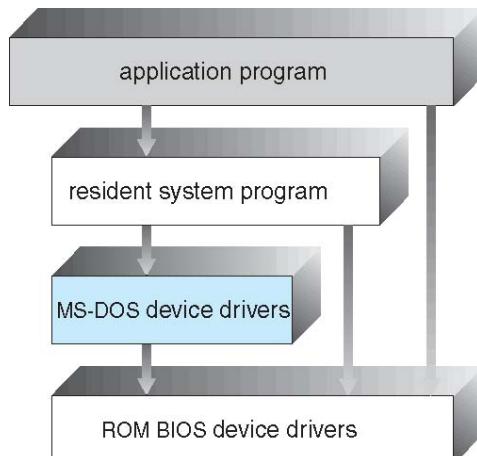
Simple Structure

- 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



Operating System Concepts – 8th Edition

2.33

Silberschatz, Galvin and Gagne ©2009



Layered Approach

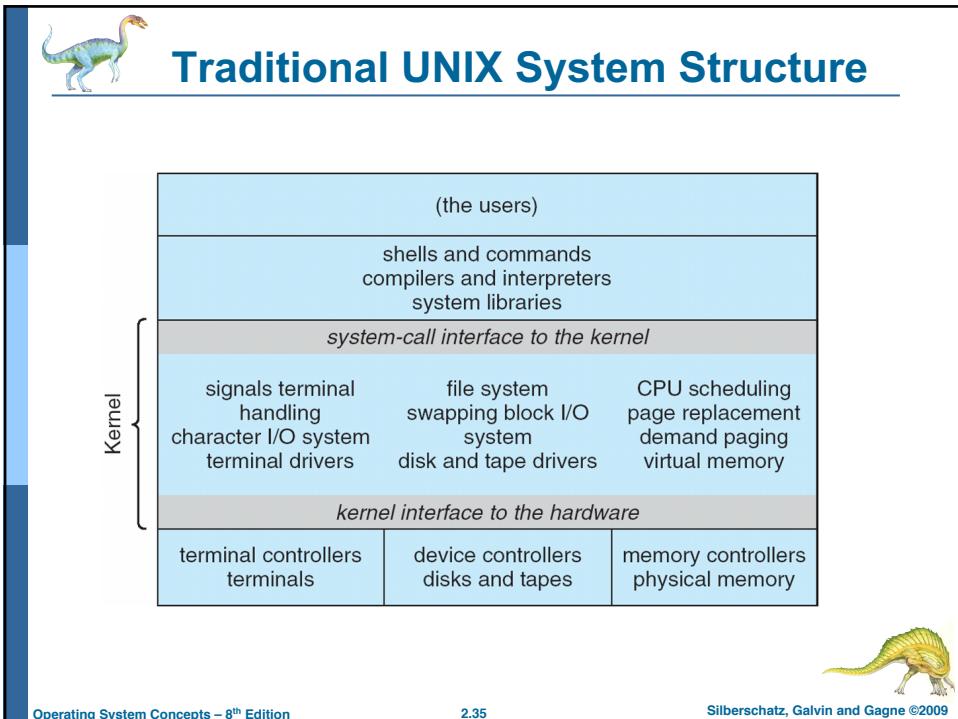
- The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level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



Operating System Concepts – 8th Edition

2.34

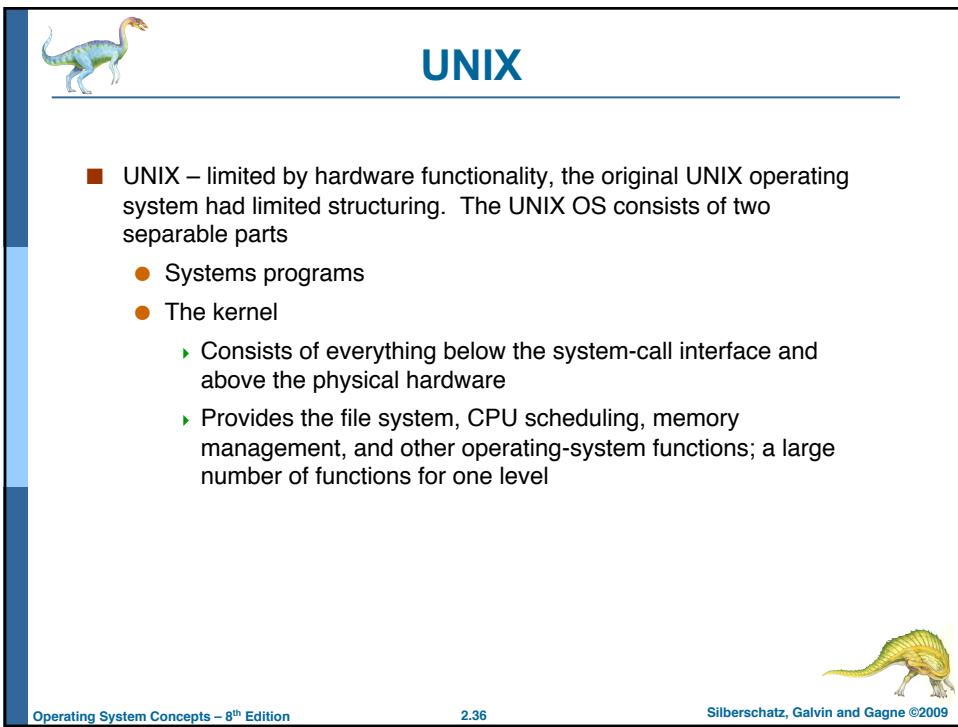
Silberschatz, Galvin and Gagne ©2009



Operating System Concepts – 8th Edition

2.35

Silberschatz, Galvin and Gagne ©2009



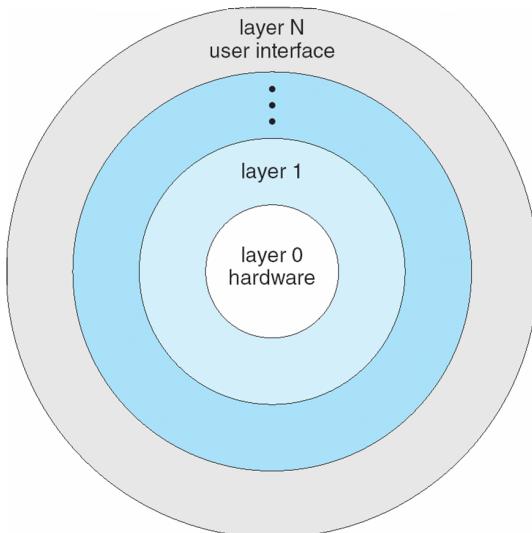
Operating System Concepts – 8th Edition

2.36

Silberschatz, Galvin and Gagne ©2009



Layered Operating System



Operating System Concepts – 8th Edition

2.37

Silberschatz, Galvin and Gagne ©2009



Microkernel System Structure

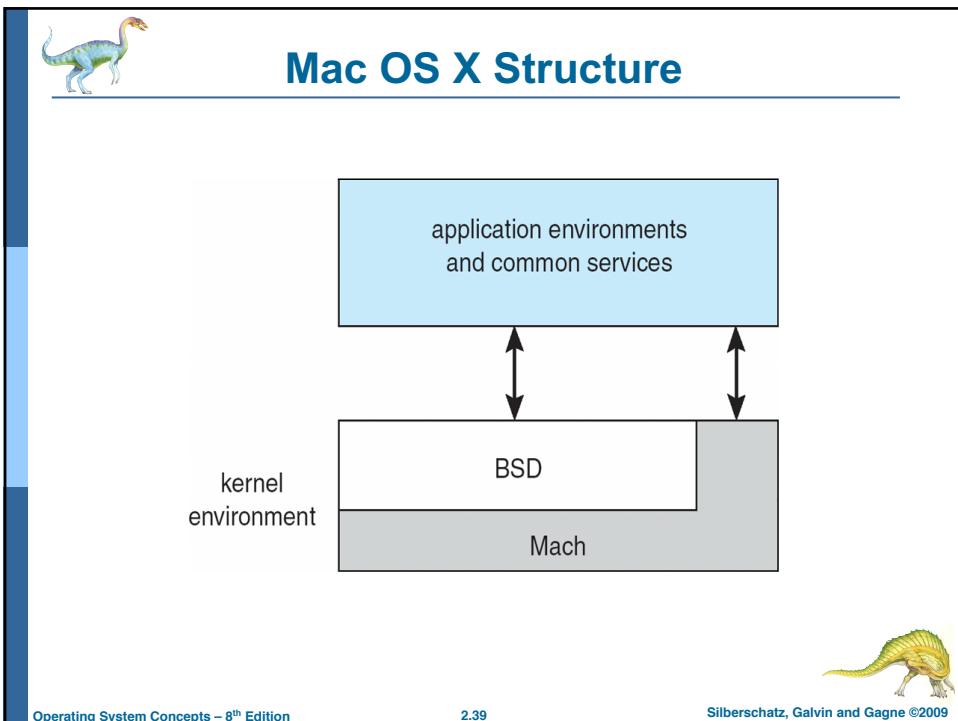
- Moves as much from the kernel into “user” space
- Communication takes place between user modules using message passing
- Benefits:
 - Easier to extend a microkernel
 - Easier to port the operating system to new architectures
 - More reliable (less code is running in kernel mode)
 - More secure
- Detriments: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



Operating System Concepts – 8th Edition

2.38

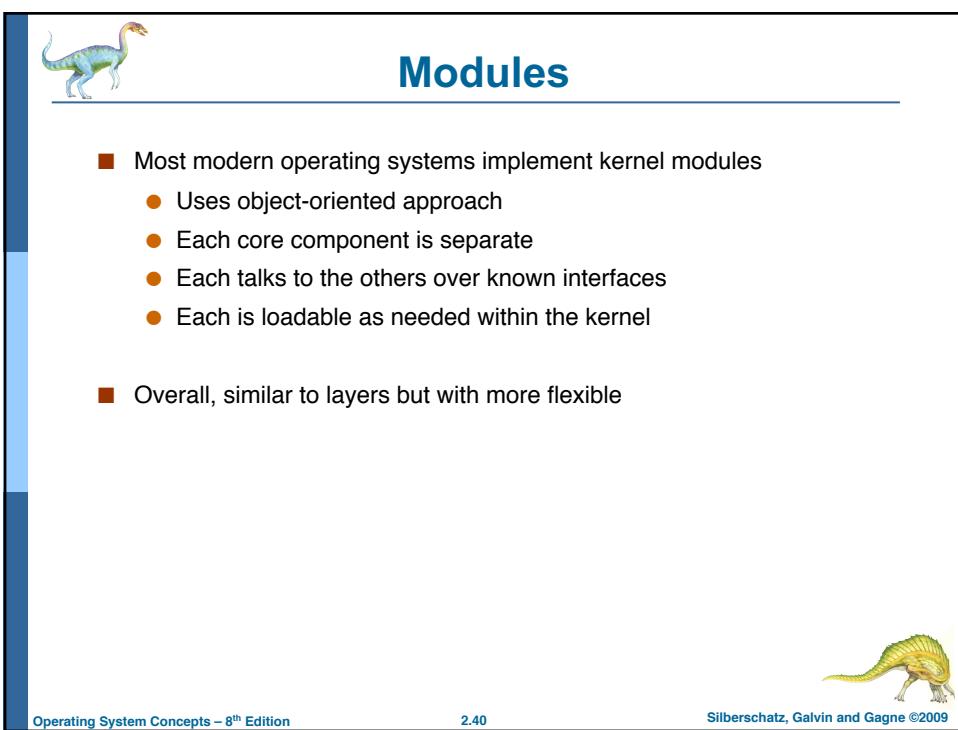
Silberschatz, Galvin and Gagne ©2009



Operating System Concepts – 8th Edition

2.39

Silberschatz, Galvin and Gagne ©2009



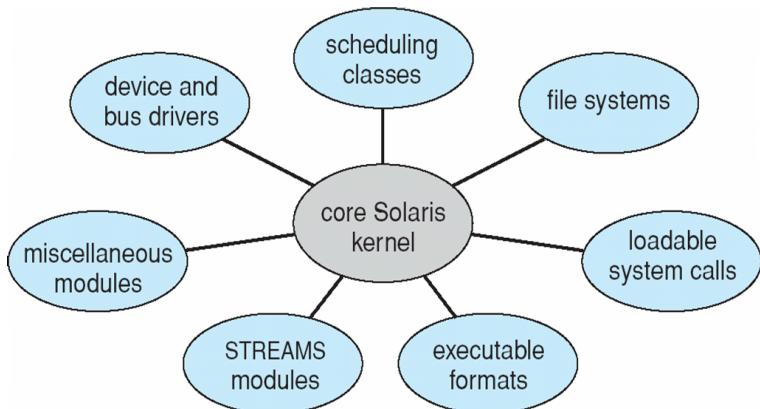
Operating System Concepts – 8th Edition

2.40

Silberschatz, Galvin and Gagne ©2009



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 **host** creates the illusion that a process has its own processor and (virtual) memory.
- Each **guest** provided with a (virtual) copy of underlying computer.



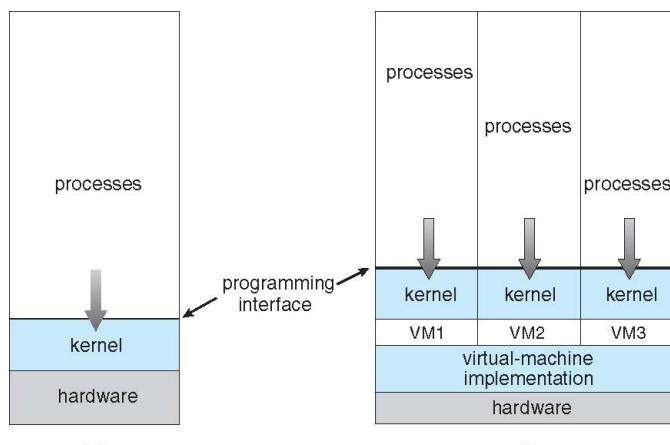


Virtual Machines History and Benefits

- First appeared commercially in IBM mainframes in 1972
- Fundamentally, multiple execution environments (different operating systems) can share the same hardware
- Protect from each other
- Some sharing of file can be permitted, controlled
- Communicate with each other, other physical systems via networking
- Useful for development, testing
- **Consolidation** of many low-resource use systems onto fewer busier systems
- “Open Virtual Machine Format”, standard format of virtual machines, allows a VM to run within many different virtual machine (host) platforms



Virtual Machines (Cont.)



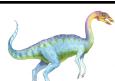
(a) Nonvirtual machine (b) virtual machine





Para-virtualization

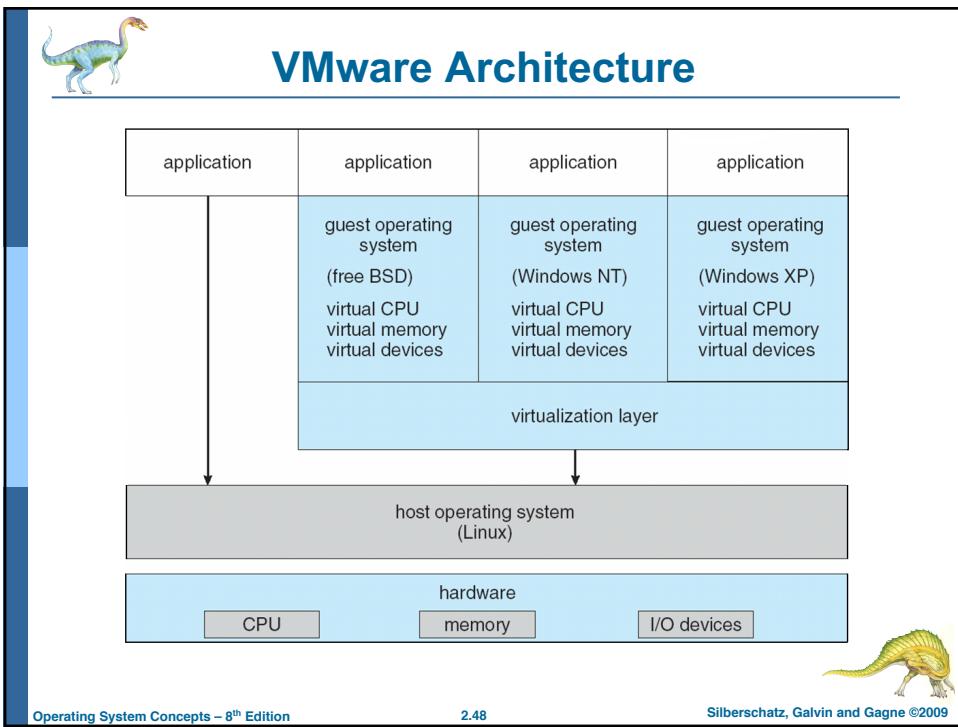
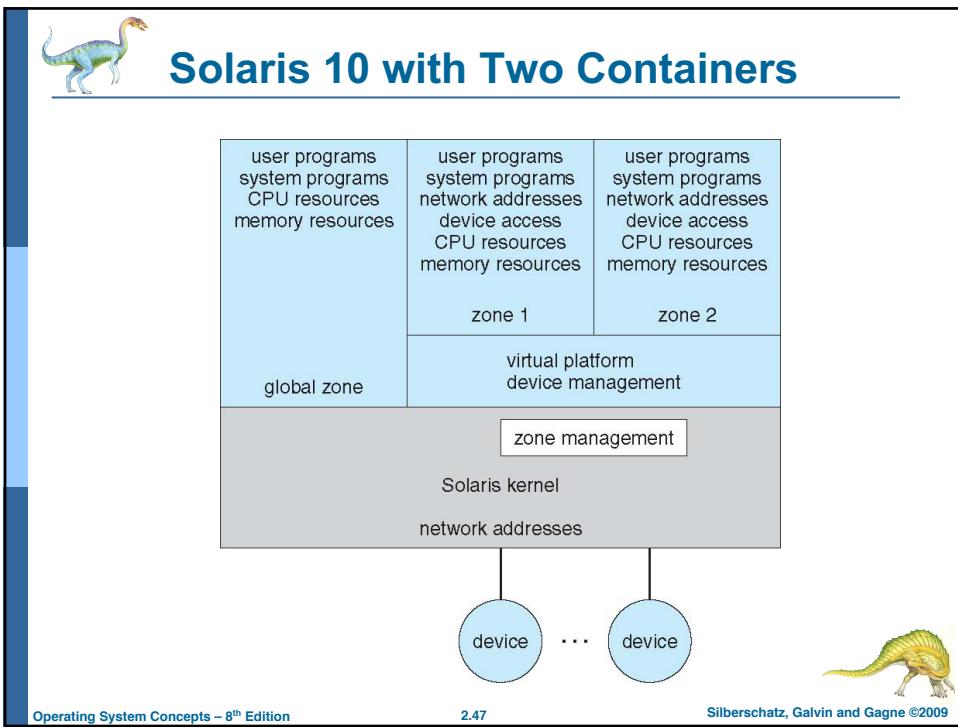
- Presents guest with system similar but not identical to hardware
- Guest must be modified to run on paravirtualized hardware
- Guest can be an OS, or in the case of Solaris 10 applications running in **containers**

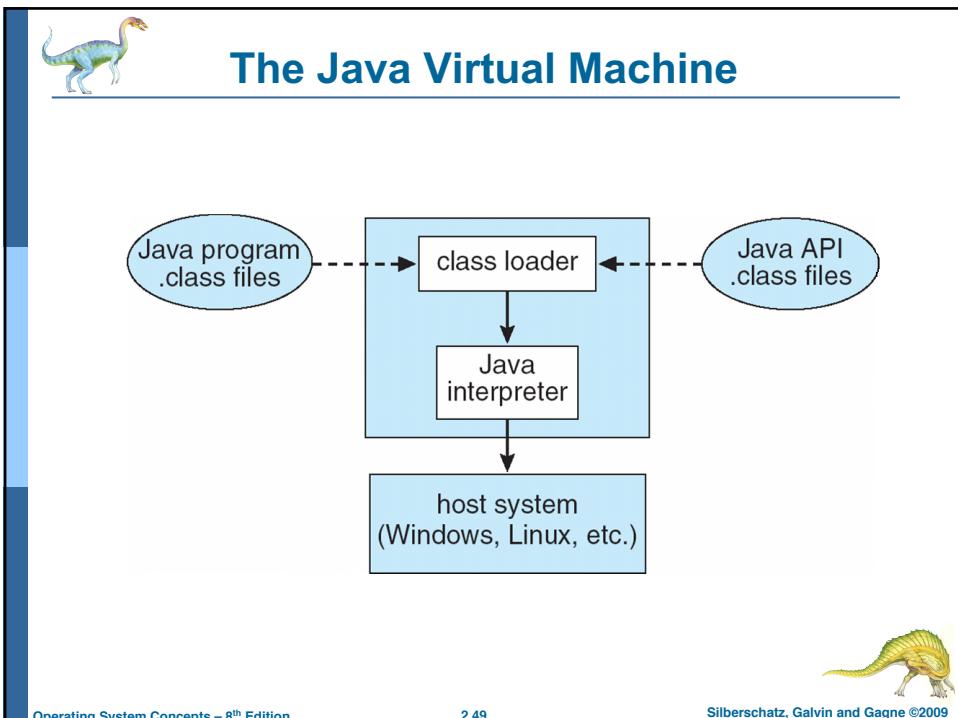


Virtualization Implementation

- Difficult to implement – must provide an *exact* duplicate of underlying machine
 - Typically runs in user mode, creates virtual user mode and virtual kernel mode
- Timing can be an issue – slower than real machine
- Hardware support needed
 - More support-> better virtualization
 - i.e. AMD provides “host” and “guest” modes



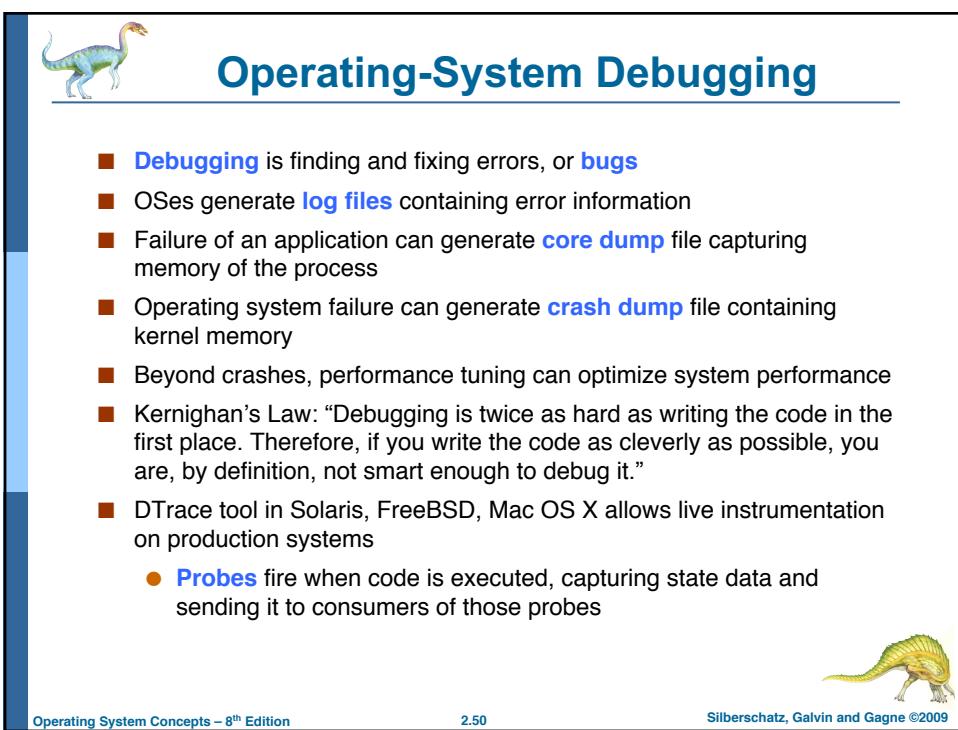




Operating System Concepts – 8th Edition

2.49

Silberschatz, Galvin and Gagne ©2009



Operating System Concepts – 8th Edition

2.50

Silberschatz, Galvin and Gagne ©2009



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 -> releaseef            K
  0 -> clear_active_fd       K
  0 <- clear_active_fd       K
  0 -> cv_broadcast          K
  0 <- cv_broadcast          K
  0 -> releaseef            K
  0 <- ioctl                 K
  0 -> ioctl                 U
  0 <- _XEventsQueued        U
  0 <- XEventsQueued          U
```



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



End of Chapter 2

