BUCKNELL UNIVERSITY
Computer Science

CSCI 315 Operating Systems Design

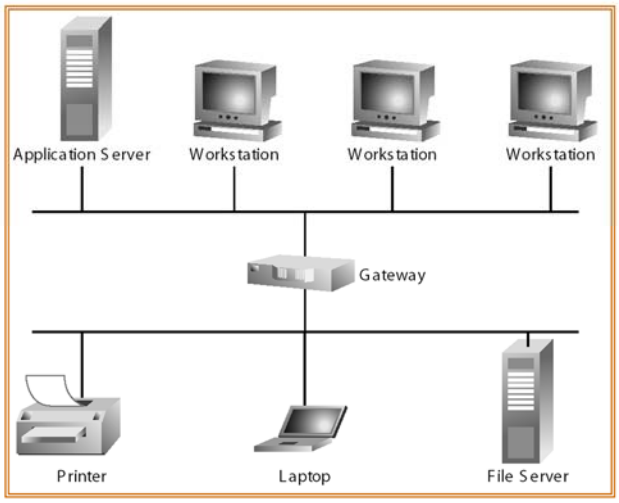
# Operating System Structures

**Notice:** The slides for this lecture have been largely based on those accompanying the seventh edition of the textbook *Operating Systems Concepts with Java*, by Silberschatz, Galvin, and Gagne (2007). Many, if not all, the illustrations contained in this presentation come from this source.

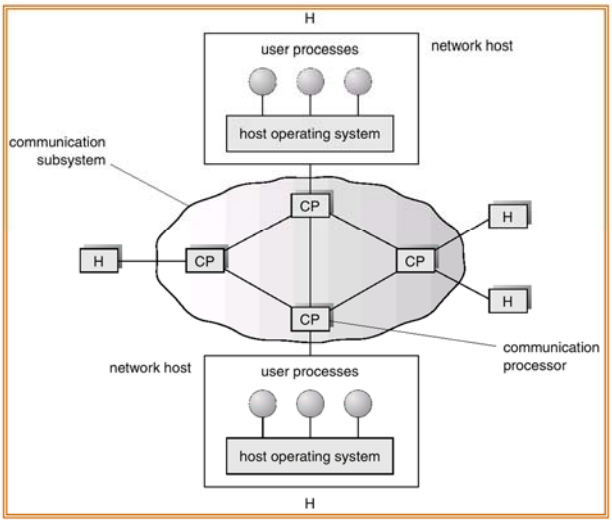
# Network Structure

- Local Area Networks (LAN)
- Wide Area Networks (WAN)

# Local Area Network Structure



# Wide Area Network Structure



# Operating System Services

- **Program execution** – system capability to load a program into memory and to run it.
- **I/O operations** – since user programs cannot execute I/O operations directly, the operating system must provide some means to perform I/O.
- **File-system manipulation** – program capability to read, write, create, and delete files.
- **Communications** – exchange of information between processes executing either on the same computer or on different systems tied together by a network. Implemented via *shared memory* or *message passing.*
- **Error detection** – ensure correct computing by detecting errors in the CPU and memory hardware, in I/O devices, or in user programs.

# Additional OS Functions

Additional functions exist not for helping the user, but rather for ensuring efficient system operations:

- **Resource allocation** – allocating resources to multiple users or multiple jobs running at the same time,
- **Accounting** – keep track of and record which users, use how much and what kinds of computer resources for account billing or for accumulating usage statistics
- **Protection** – ensuring that all access to system resources is controlled.

# Hardware Protection

- Dual-Mode Operation
- I/O Protection
- Memory Protection
- CPU Protection

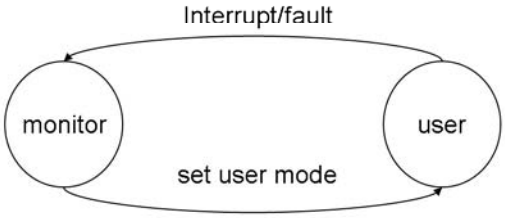
# Dual-Mode Operation

- Sharing system resources requires operating system to ensure that an incorrect program or poorly behaving human cannot cause other programs to execute incorrectly.

- OS must provide hardware support to differentiate between at least two modes of operations:
  1. ***User mode*** – execution done on behalf of a user,
  2. ***Monitor mode*** (also *kernel mode* or *system mode*) – execution done on behalf of operating system.

# Dual-Mode Operation (Cont.)

- ***Mode bit*** added to computer hardware to indicate the current mode: monitor (0) or user (1).
- When an interrupt or fault occurs hardware switches to monitor mode.




***Privileged instructions*** can be issued only in monitor mode.

# I/O Protection

- All I/O instructions are privileged instructions.
- Must ensure that a user program could never gain control of the computer in monitor mode (i.e., a user program that, as part of its execution, stores a new address in the interrupt vector).

# CPU Protection

- A ***timer*** interrupts the computer after a specified period to ensure the operating system maintains control:
  - Timer is decremented every clock tick,
  - When timer reaches the value 0, an interrupt occurs.
- Timer commonly used to implement time-sharing.
- Timer also used to compute the current time.
- Load-timer is a privileged instruction.

# Common OS Components

- Process Management
- Main Memory Management
- File Management
- I/O System Management
- Secondary-Storage Management
- Networking
- Protection System
- Command-Interpreter System

# Command-Interpreter System

Many commands are given to the operating system by control statements which deal with:

- Process creation and management,
- I/O handling,
- Secondary-storage management,
- Main-memory management,
- File-system access,
- Protection,
- Networking.

# Command-Interpreter System

The program that reads and interprets control statements is called variously:

- *command-line interpreter,* or
- *shell (in UNIX).*

Its function is to read in and execute the next command statement.

# General-System Architecture

- Given the I/O instructions are privileged, how does the user program perform I/O?
- System call – the method used by a process to request action by the operating system:
  - Usually takes the form of a trap to a specific location in the interrupt vector,
  - Control passes through the interrupt vector to a service routine in the OS, and the mode bit is set to monitor mode,
  - The monitor verifies that the parameters are correct and legal, executes the request, and returns control to the instruction following the system call.

# System Calls

- ***System calls*** provide the interface between a running program and the operating system:
  - Generally available as assembly-language instructions,
  - Languages defined to replace assembly language for systems programming allow system calls to be made directly (e.g., C, C++).
- Three general methods are used to *pass parameters* between a running program and the operating system:
  - Pass parameters in *registers,*
  - *Push* (store) the parameters onto the *stack* by the program, and *pop* off the stack by operating system,
  - Store the parameters in a table in memory, and the table address is passed as a parameter in a register.

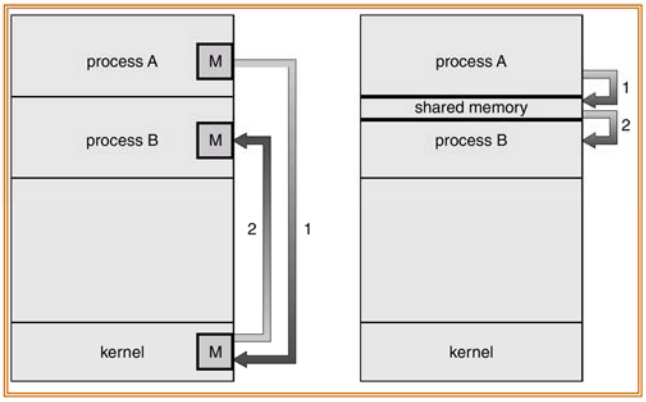
# Types of System Calls

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

# 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.

# Communication Models




Message Passing

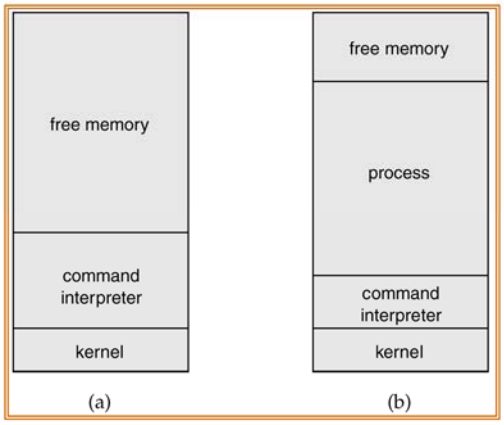
Shared Memory

# MS-DOS System 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.

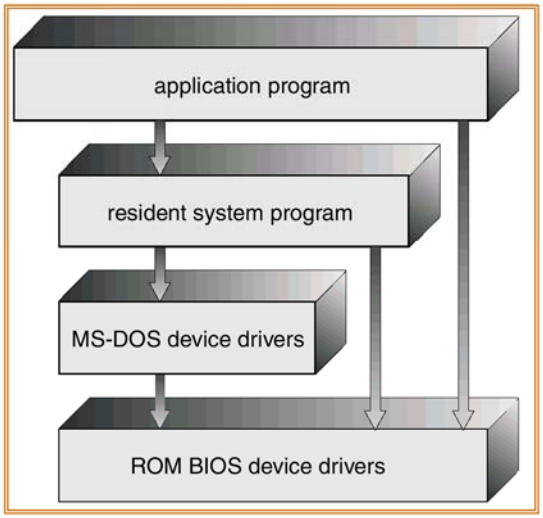
# MS-DOS Execution




At System Start-up

Running a Program

# MS-DOS Layer Structure

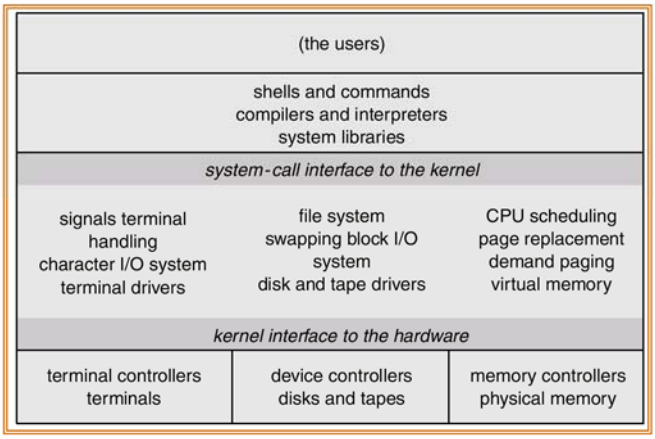


# UNIX System Structure

UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring. The UNIX OS consists of two separable parts:

- Systems programs, and
- 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.

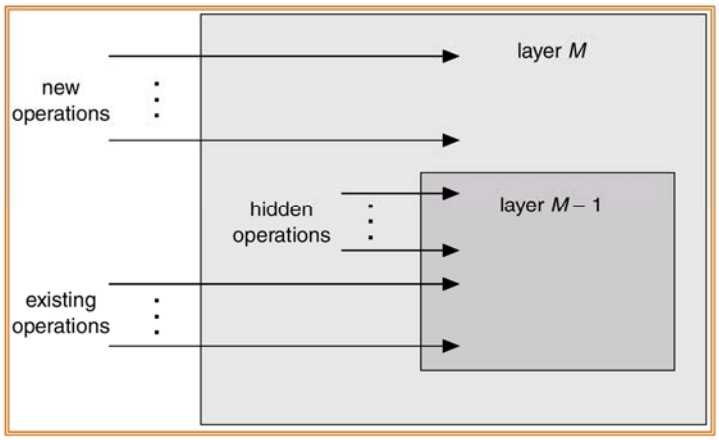
# UNIX System Structure



# Layered 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.

# An Operating System Layer



# 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.

# 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, and
  - Each is loadable as needed within the kernel.
- Overall, similar to layers but with more flexibility.

# Virtual Machines

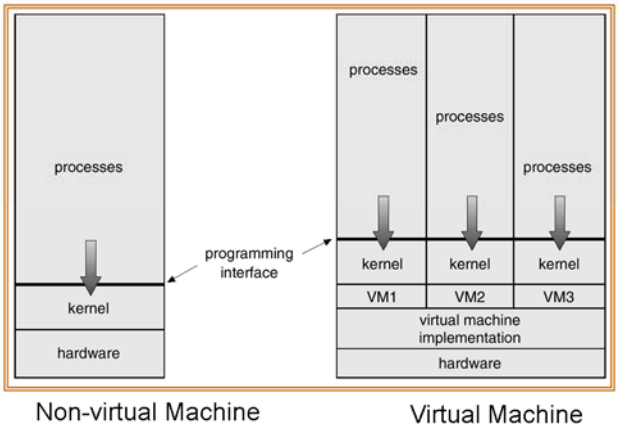
- A ***virtual machine*** takes the layered approach to its logical conclusion. It treats hardware and the operating system kernel as though they were all hardware.
- A virtual machine provides an interface *identical* to the underlying bare hardware.
- The operating system creates the illusion of multiple 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



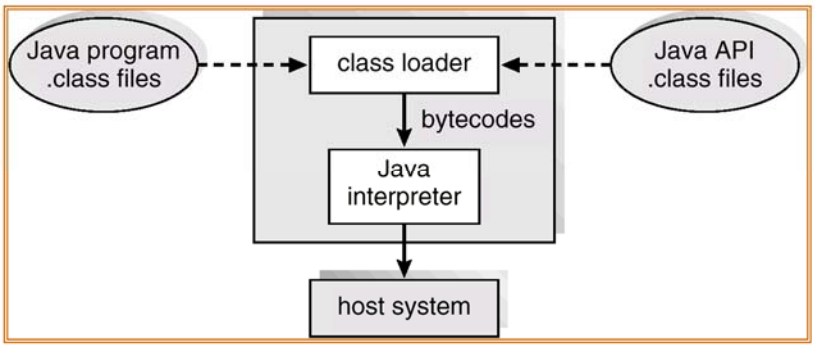
# [Ad|Disad]vantages of Virtual Machines

- 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.

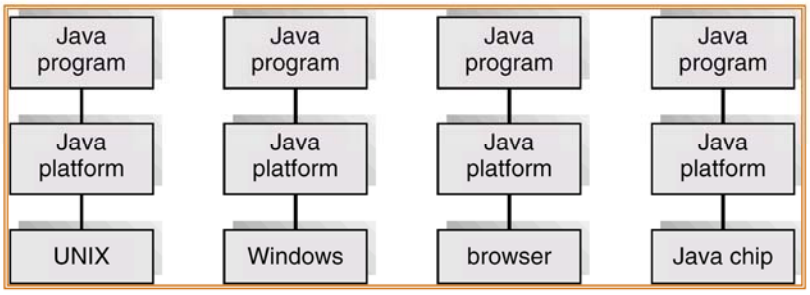
# Java Virtual Machine

- Compiled Java programs are platform-neutral bytecodes executed by a Java Virtual Machine (JVM).
- JVM consists of:
  - Class loader,
  - Class verifier,
  - Runtime interpreter.
- Just-In-Time (JIT) compilers increase performance.

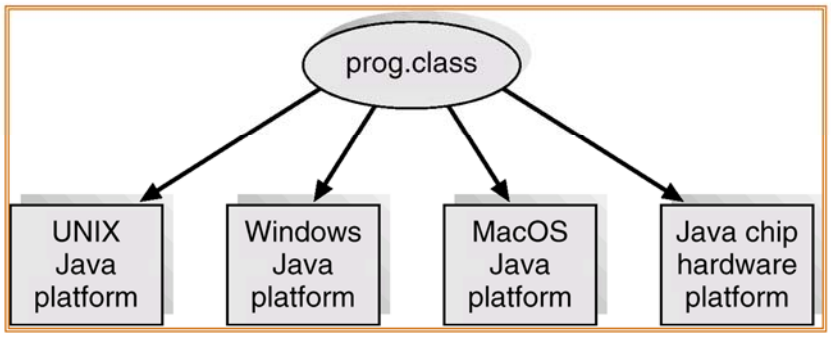
# The Java Virtual Machine



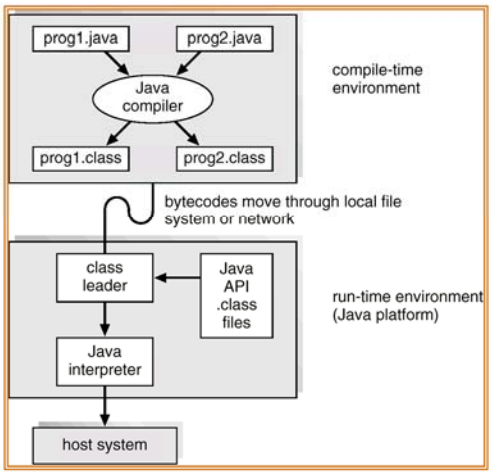
# The Java Platform



# Java .class File on Cross Platforms



# Java Development Environment



# Operating System Design Goals

- User goals – operating system should be convenient to use, easy to learn, reliable, secure, and fast.
- System goals – operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient.

# System Implementation

- Traditionally written in assembly language, operating systems can now be written in higher-level languages.
- Code written in a high-level language:
  - Can be written faster,
  - Is more compact, and
  - Is easier to understand and debug.
- An operating system is far easier to *port* (move to some other hardware) if it is written in a high-level language.