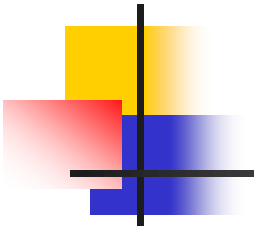
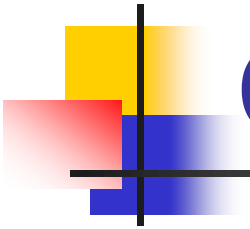
Chapter 3: Operating-System Structures

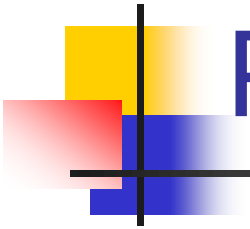


- System Components
- Operating System Services
- System Calls
- System Programs
- System Structure
- Virtual Machines
- System Design and Implementation
- System Generation



Common System Components

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



Review Questions

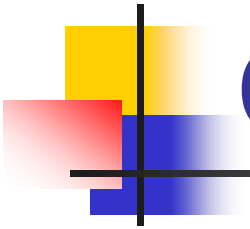
- **Evaluate** the trade-offs between **monolithic**, **microkernel**, and **modular** OS architectures. Which design is most appropriate for real-time embedded systems, and why?
- An operating system designer must choose between implementing **preemptive** or **non-preemptive** multitasking. **Judge** which is better for a general-purpose desktop OS and **justify** your recommendation.
- **Assess** the importance of **separation between user space and kernel space** in OS design. What are the implications for system stability and security if this principle is not followed?
- **Critique** the decision to tightly integrate device drivers into the kernel (as in traditional Unix) versus loading them dynamically (as in modern Linux). What are the pros and cons of each approach?

Review Questions

- **Evaluate** how well the **layered architecture** of an OS supports maintainability and extensibility. Can too many layers cause performance issues? Justify your answer.
- **Compare and evaluate** the effectiveness of **cooperative multitasking** and **preemptive multitasking** in terms of responsiveness and fairness in OS process management.
- A team is designing an operating system for a cloud platform. **Evaluate** the relative importance of scalability, portability, and modularity in the system's architecture. Which should be prioritized, and why?

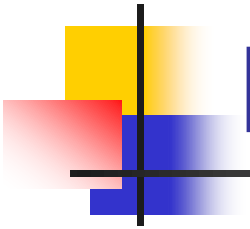
Review Questions

- **Judge** the role of **interrupt handling** mechanisms in the performance of an operating system. What design choices could degrade responsiveness or throughput?
- **Critically assess** the inclusion of **virtual memory** management in the OS kernel. What are the performance trade-offs, and when might a system benefit from omitting this feature?
- **Evaluate** the implementation of **system calls** as the primary interface between user applications and kernel services. Are there more secure or efficient alternatives in OS design? Justify your position.



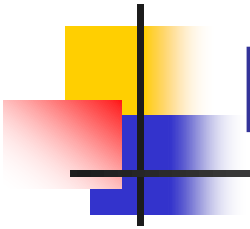
Common System Components

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



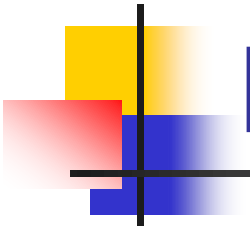
Process Management

- A *process* is a program in execution. A process needs certain resources, including CPU time, memory, files, and I/O devices, to accomplish its task.
- The operating system is responsible for the following activities in connection with process management.
  - Process creation and deletion.
  - process suspension and resumption.
  - Provision of mechanisms for:
    - process synchronization
    - process communication



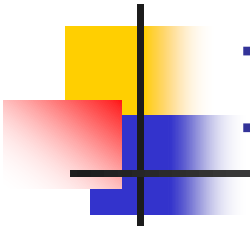
Main-Memory Management

- Memory is a large array of words or bytes, each with its own address. It is a repository of quickly accessible data shared by the CPU and I/O devices.
- Main memory is a volatile storage device. It loses its contents in the case of system failure.
- The operating system is responsible for the following activities in connections with memory management:
  - Keep track of which parts of memory are currently being used and by whom.
  - Decide which processes to load when memory space becomes available.
  - Allocate and deallocate memory space as needed.



File Management

- A file is a collection of related information defined by its creator. Commonly, files represent programs (both source and object forms) and data.
- The operating system is responsible for the following activities in connections with file management:
  - File creation and deletion.
  - Directory creation and deletion.
  - Support of primitives for manipulating files and directories.
  - Mapping files onto secondary storage.
  - File backup on stable (nonvolatile) storage media.



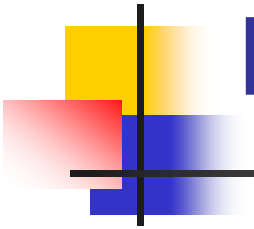
I/O System Management

- The I/O system consists of:
  - A buffer-caching system
  - A general device-driver interface
  - Drivers for specific hardware devices

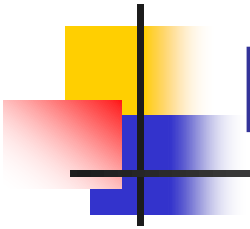
Secondary-Storage Management

- Since main memory (*primary storage*) is volatile and too small to accommodate all data and programs permanently, the computer system must provide *secondary storage* to back up main memory.
- Most modern computer systems use disks as the principle on-line storage medium, for both programs and data.
- The operating system is responsible for the following activities in connection with disk management:
  - Free space management
  - Storage allocation
  - Disk scheduling

Networking (Distributed Systems)

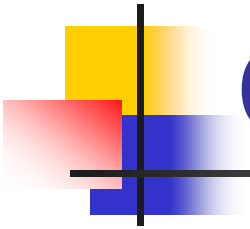


- A *distributed* system is a collection processors that do not share memory or a clock. Each processor has its own local memory.
- The processors in the system are connected through a communication network.
- Communication takes place using a *protocol.*
- A distributed system provides user access to various system resources.
- Access to a shared resource allows:
  - Computation speed-up
  - Increased data availability
  - Enhanced reliability



Protection System

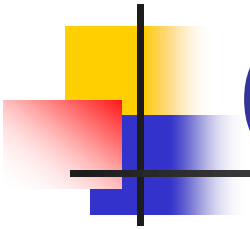
- *Protection* refers to a mechanism for controlling access by programs, processes, or users to both system and user resources.
- The protection mechanism must:
  - distinguish between authorized and unauthorized usage.
  - specify the controls to be imposed.
  - provide a means of enforcement.



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 (Cont.)



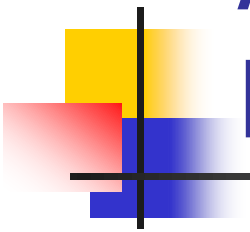
- The program that reads and interprets control statements is called variously:
  - command-line interpreter
  - shell (in UNIX)

Its function is to get and execute the next command statement.

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 Operating System Functions



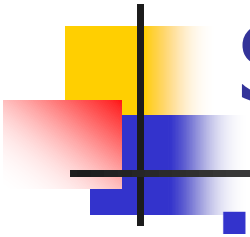
Additional functions exist not for helping the user, but rather for ensuring efficient system operations.

- 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.

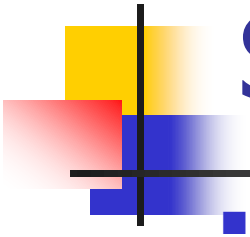
System Calls

- Is programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on.
- The interface between a process and an operating system is provided by system calls
- In general, system calls are available as assembly language instructions
- System calls are usually made when a process in user mode requires access to a resource. Then it requests the kernel to provide the resource via a system call

System Calls – When required?



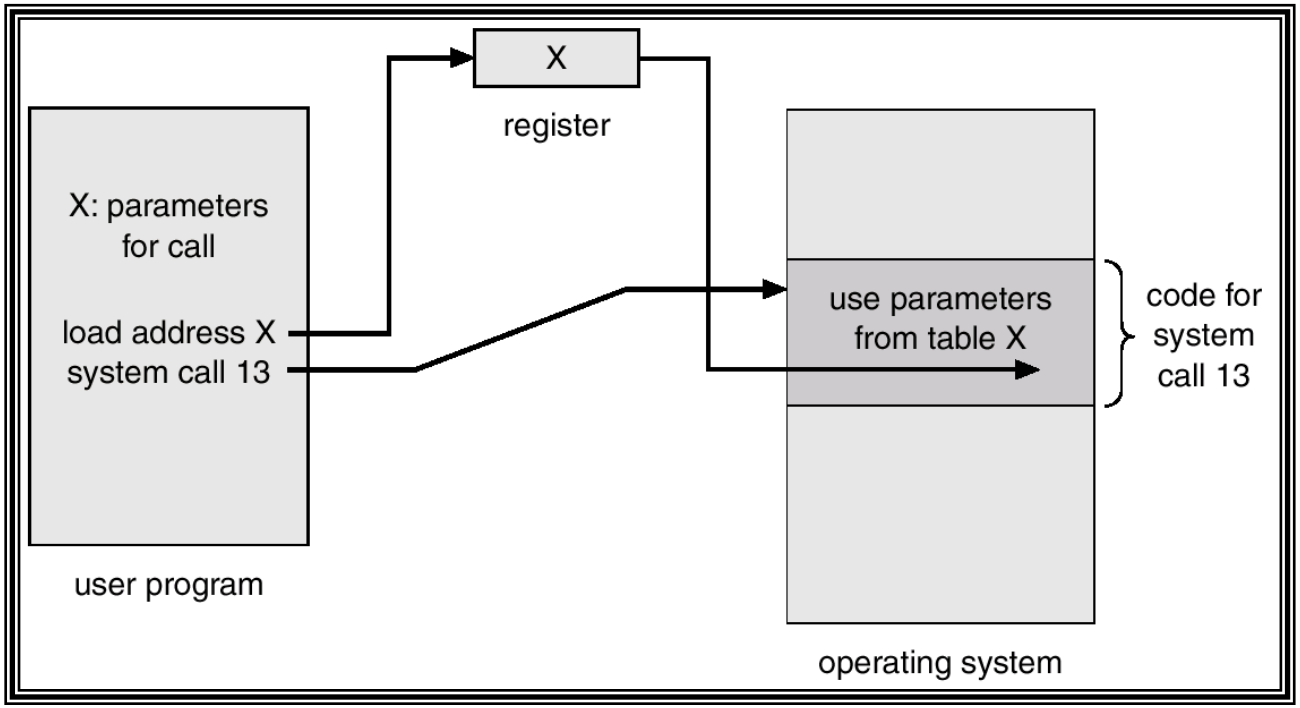
- If a file system requires the creation or deletion of files. Reading and writing from files also require a system call
- Creation and management of new processes
- Network connections also require system calls. This includes sending and receiving packets.
- Access to a hardware devices such as a printer, scanner etc. requires a system call.

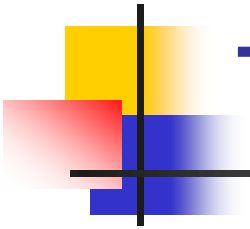


System Calls

- 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*.
  - Store the parameters in a table in memory, and the table address is passed as a parameter in a register.
  - *Push* (store) the parameters onto the *stack* by the program, and *pop* off the stack by operating system.

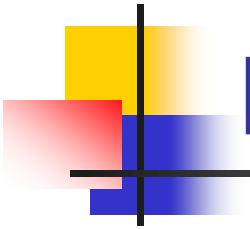
Passing of Parameters As A Table





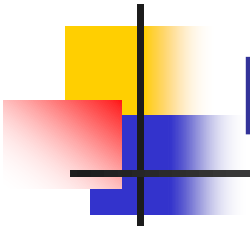
Types of System Calls

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



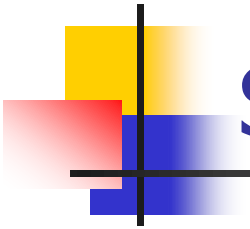
Process Control System Calls

- Deal with processes such as create, end, abort, terminate, allocate and free memory, etc.
- Functions:
  - End and Abort
  - Load and Execute
  - Create Process and Terminate Process
  - Wait and Signed Event
  - Allocate and free memory



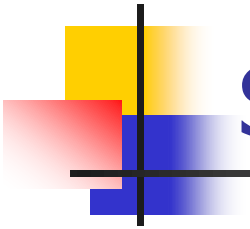
File Management System Calls

- Responsible for file manipulation such as creating a file, reading a file, writing into a file etc.
- Functions:
  - Create a file
  - Delete file
  - Open and close file
  - Read, write, and reposition
  - Get and set file attribute



System Calls – Device and Info

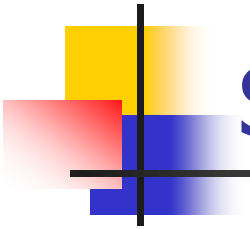
- Device Management: Responsible for device manipulation such as reading from device buffers, writing into device buffers etc.
- Functions:
  - Request and release device
  - Logically attach/ detach devices
  - Get and Set device attributes
- Information Maintenance: Handle information and its transfer between the operating system and the user program
- Functions:
  - Get or set time and date
  - Get process and device attributes



System Calls – Communication

- Communication: These system calls are useful for inter process communication. They also deal with creating and deleting a communication connection.

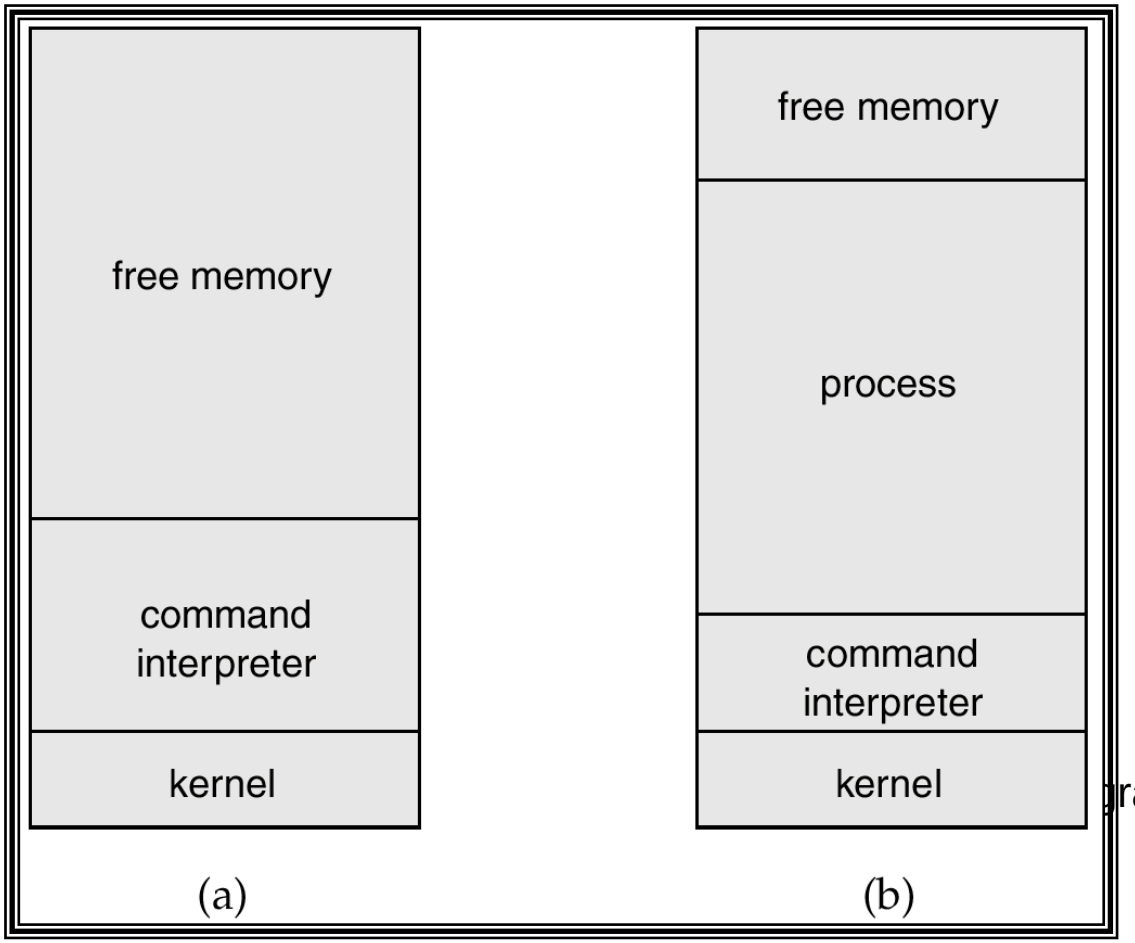
- Functions:
  - Create, delete communications connections
  - Send, receive message
  - Help OS to transfer status information
  - Attach or detach remote devices



System Calls – Protection & Security

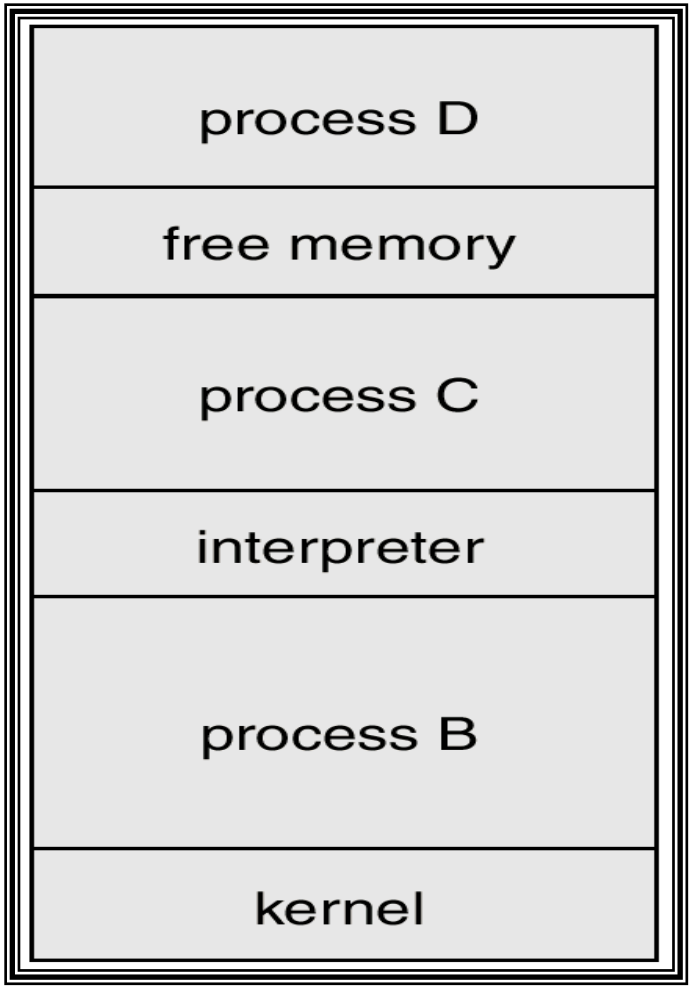
- Protection:
  - When several separate processes execute concurrently , it should not be possible for one process to interfere with another (one process execution should not interrupt another process) – *Interference reduction*
  - Each process should access the system resources in control(resources should not be wasted unnecessarily).
- Security:
  - Security for system from outsiders is also important
  - So each user has to authenticate self to the system
  - By using password to (Access the system resources)

MS-DOS Execution



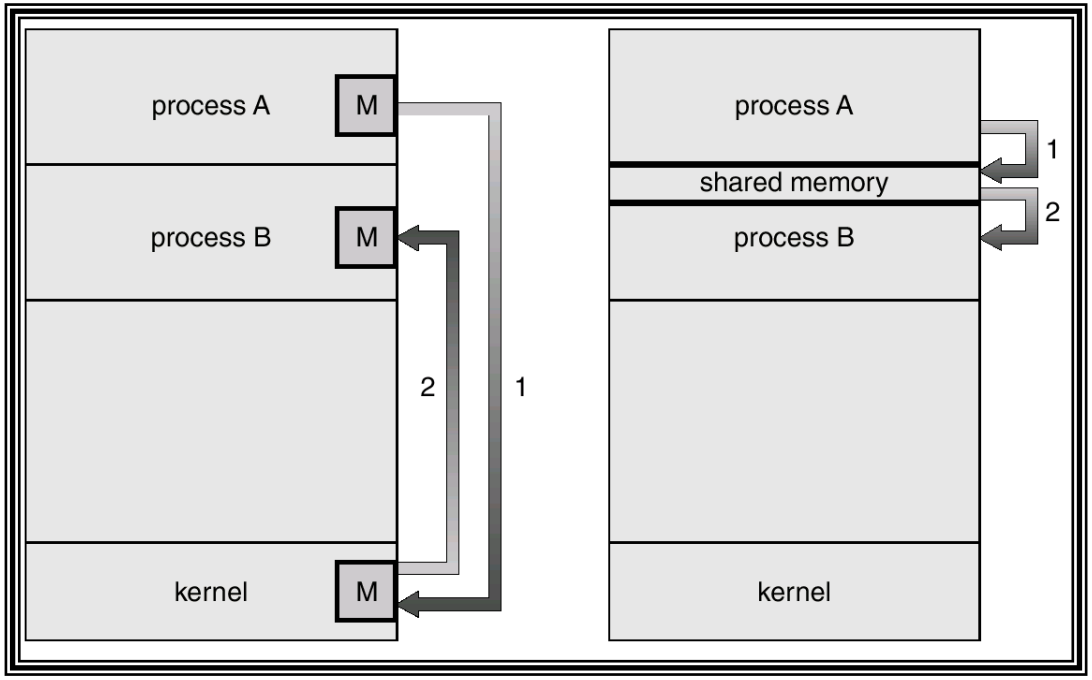
gram

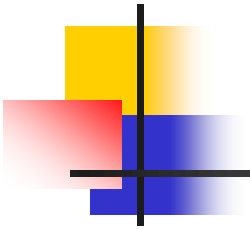
UNIX Running Multiple Programs



Communication Models

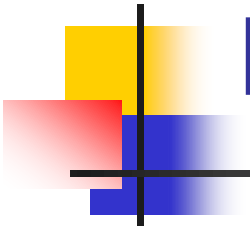
Communication may take place using either message passing or shared memory.





System Programs

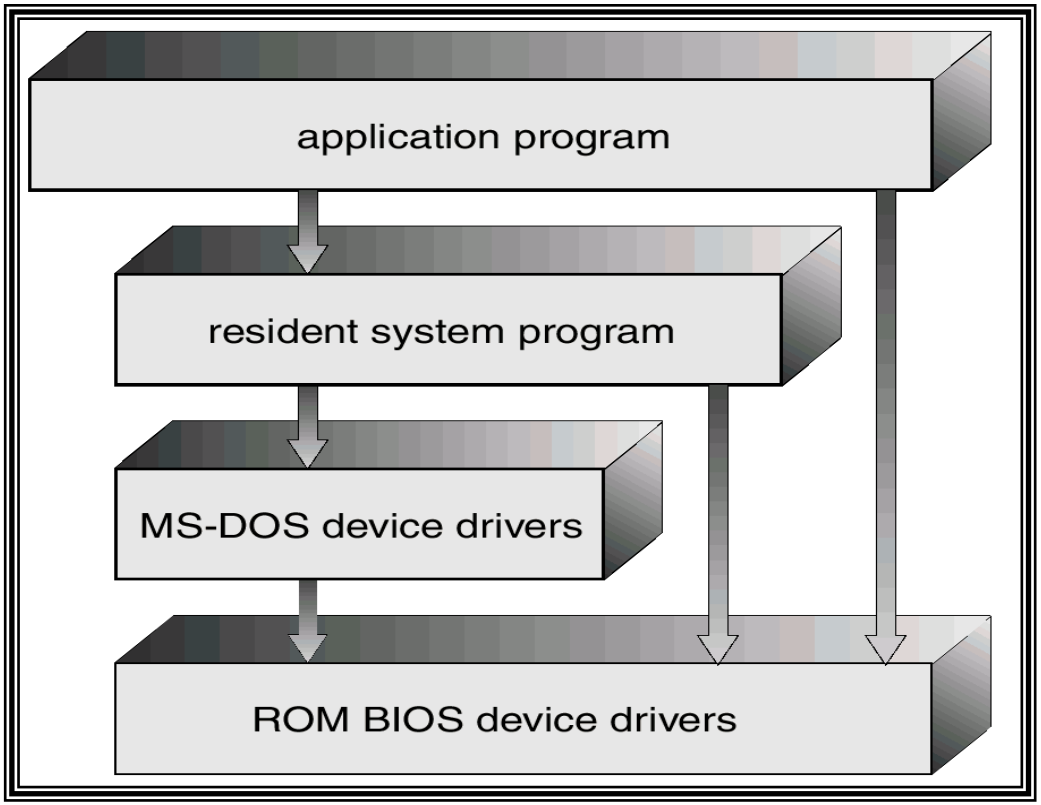
- System programs provide a convenient environment for program development and execution. The can be divided into:
  - File manipulation
  - Status information
  - File modification
  - Programming language support
  - Program loading and execution
  - Communications
  - Application programs
- Most users' view of the operation system is defined by system programs, not the actual system calls.



MS-DOS System Structure

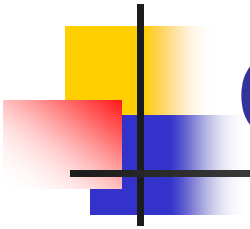
- MS-DOS – written to provide the most functionality in the least space
  - not divided into modules
  - Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated

MS-DOS 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
  - 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.



OS: Design and Implementation

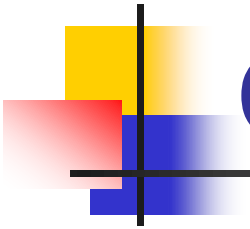
- Operating system can be implemented with the help of various structures.
- The structure of the OS depends mainly on how the various common components of the operating system are interconnected and melded into the kernel.
- Depending on this we have following structures of the operating system:

OS: Design and Implementation

- 1.Simple structure:
  - Such operating systems do not have well defined structure and are small, simple and limited systems.
  - The interfaces and levels of functionality are not well separated.
  - MS-DOS is an example of such operating system. In MS-DOS application programs are able to access the basic I/O routines.
  - These types of operating system cause the entire system to crash if one of the user programs fails. Diagram of the structure of MS-DOS is shown below

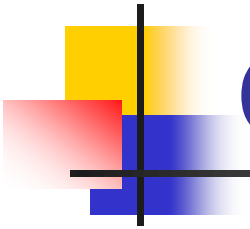
OS: Design and Implementation

- 2. Monolithic structure:
  - The monolithic operating system is a very basic operating system in which file management, memory management, device management, and process management is directly controlled within the kernel.
    - All these components like file management, memory management etc. are located within the kernel.



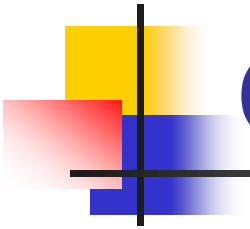
OS: Design and Implementation

- 2. Monolithic structure:
  - Functionality of the OS is invoked with simple function calls within the kernel, which is one large program
  - Device drivers are loaded into the running kernel and become part of the kernel.



OS: Design and Implementation

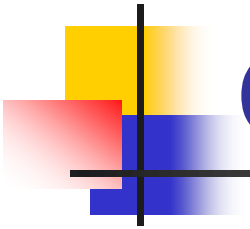
- Advantages:
  - Simple structure: This type of operating system has a simple structure. All the components needed for processing are embedded into the kernel.
  - Works for smaller tasks: It works better for performing smaller tasks as it can handle limited resources.
  - Communication between components: All the components can directly communicate with each other and also with the kernel.
  - Fast operating system: The code to make monolithic kernel is very fast and robust



OS: Design and Implementation

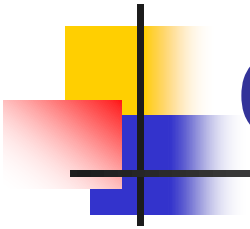
- Disadvantages:
  - Code written in this operating system (OS) is difficult to port.
  - Monolithic OS has more tendency to generate errors and bugs. The reason is that user processes use same address locations as the kernel.
  - Adding and removing features from monolithic OS is very difficult. All the code needs to be rewritten and recompiled to add or remove any feature.

  Examples: VMS, Linux, OS/360, OpenVMS, Multics, AIX, BSD



OS: Design and Implementation

- **Layered Structure**
  - An OS can be broken into pieces and retain much more control on system.
  - In this structure the OS is broken into number of layers (levels).
  - The bottom layer (layer 0) is the hardware and the topmost layer (layer N) is the user interface.
  - These layers are so designed that each layer uses the functions of the lower level layers only



OS: Structure

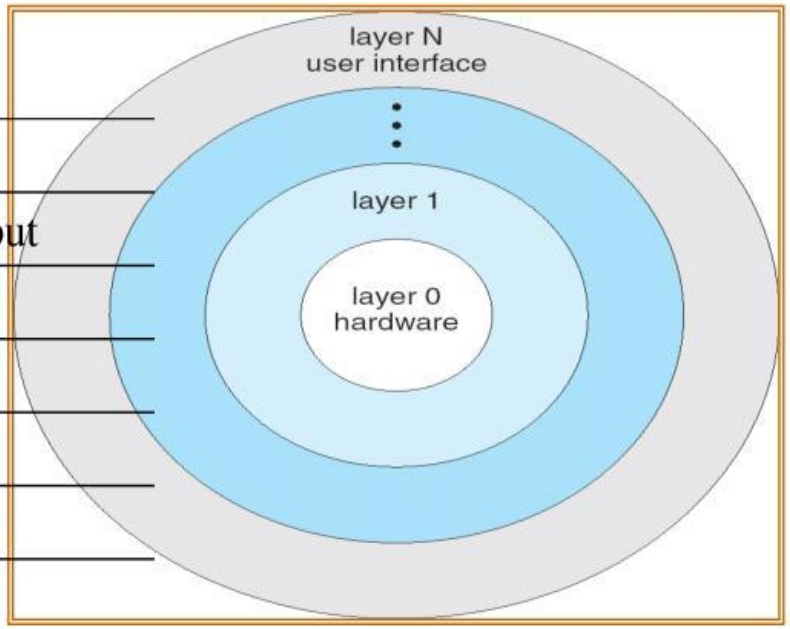
- **Layered Structure (Cont.)**
  - This simplifies the debugging process as if lower level layers are debugged and an error occurs during debugging then the error must be on that layer only as the lower level layers have already been debugged.
  - The main disadvantage of this structure is that at each layer, the data needs to be modified and passed on which adds overhead to the system
  - Moreover careful planning of the layers is necessary as a layer can use only lower level layers. UNIX is an example of this structure.

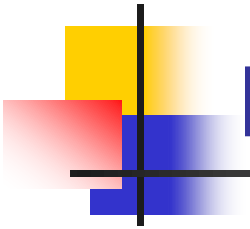
Layered Structure of the THE OS

- A layered design was first used in THE operating system.

Its six layers are as follows:

layer 5:	user programs
layer 4:	buffering for input and output
layer 3:	Process management
layer 2:	memory management
layer 1:	CPU scheduling
layer 0:	hardware





Micro-Kernel Operating System:

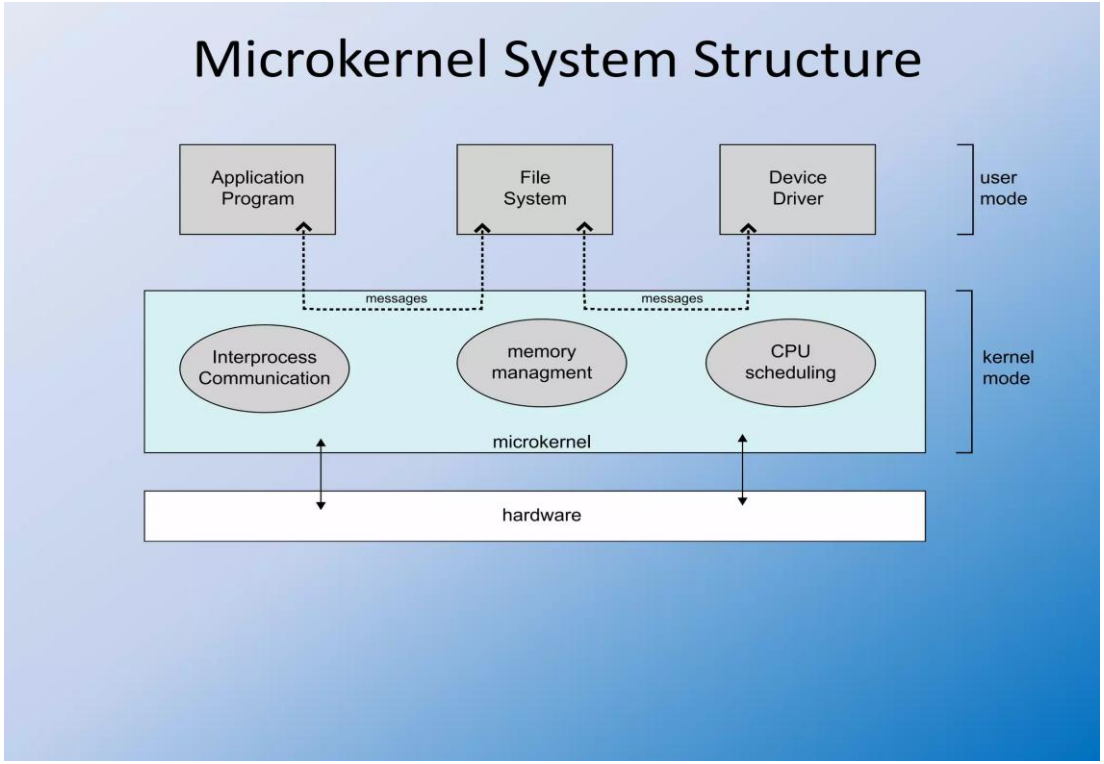
- Designs operating system by removing all non-essential components from the kernel, implementing them as system and user programs. Result: smaller kernel called the micro-kernel
- Moves as much from the kernel into user space
- Communication takes place between user modules using message passing
- Thus it is more secure and reliable as if a service fails then rest of the operating system remains untouched.
- Mac OS is an example of this type of OS.

Micro-Kernel Operating System:

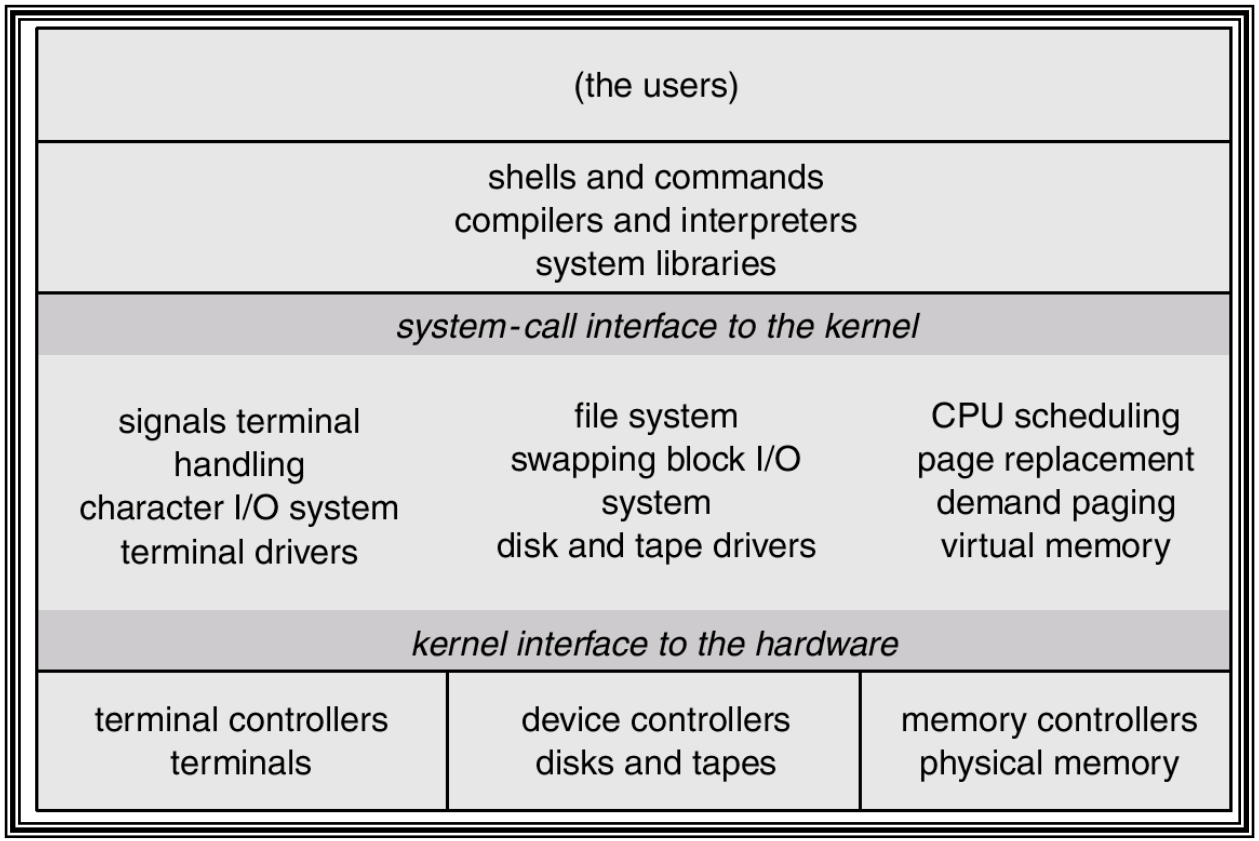
- **Kernel**: Part of an OS that manages system resources.
  - Acts as a bridge btn software and hardware of the computer.
  - Is one of the first program which is loaded on start-up after the bootloader.
  - The Kernel is also responsible for offering secure access to the machine's hardware for various programs. It also decides when and how long a certain application uses specific hardware
- Advantages:
  - Easier to port the operating system to new architectures
  - All new services need to be added to user space and does not require the kernel to be modified 3. More secure
  - More reliable (less code is running in kernel mode)
  - Kernel architecture is small and isolated hence can function better.
  - Expansion of the system is easier, it is simply added in the system application without disturbing the kernel

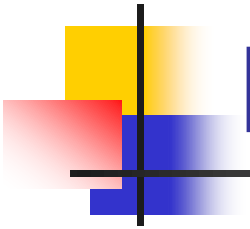
Micro-Kernel Operating System:

- Disadvantages:
  - Performance overhead of user space to kernel space
- Example: Mac OS, Eclipse IDE



UNIX System Structure

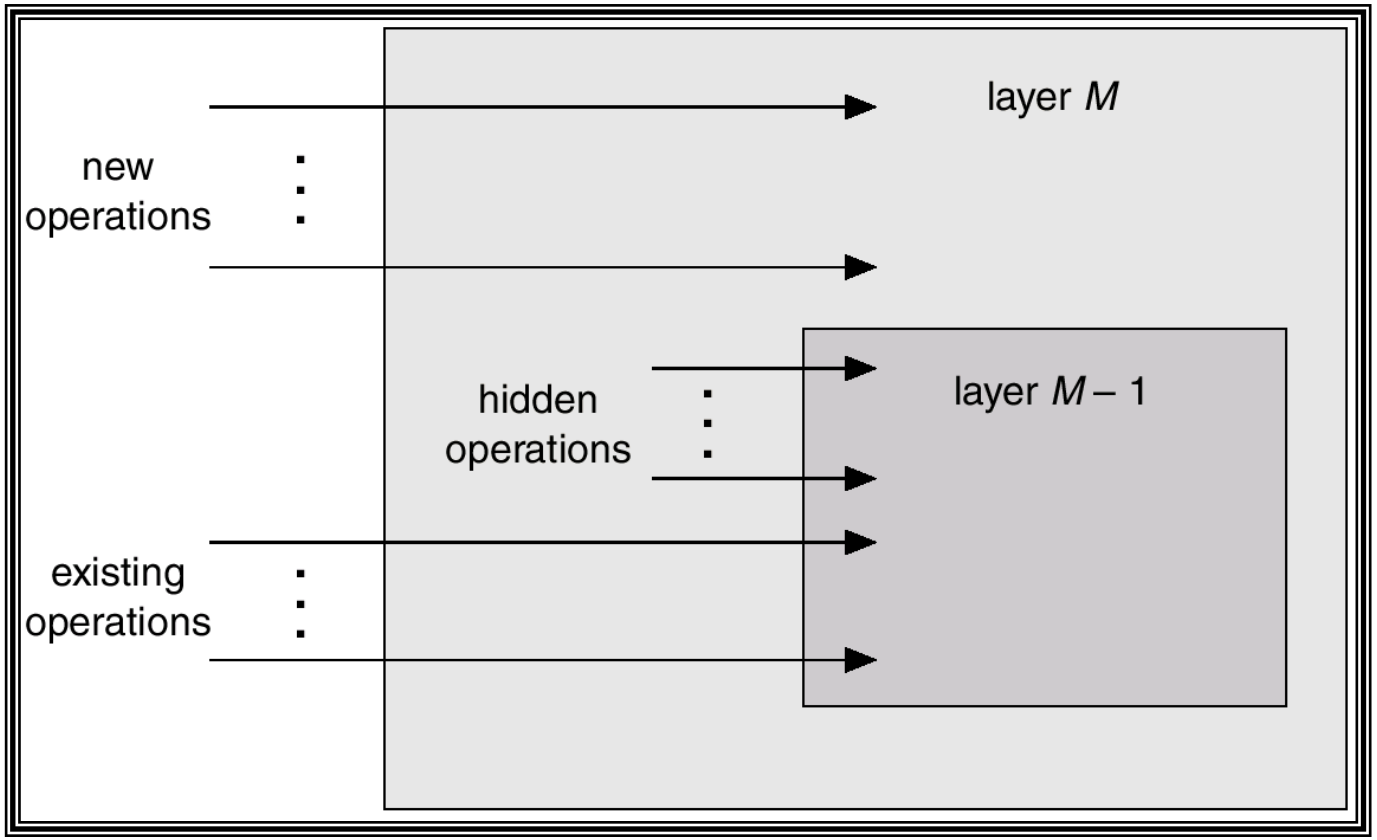




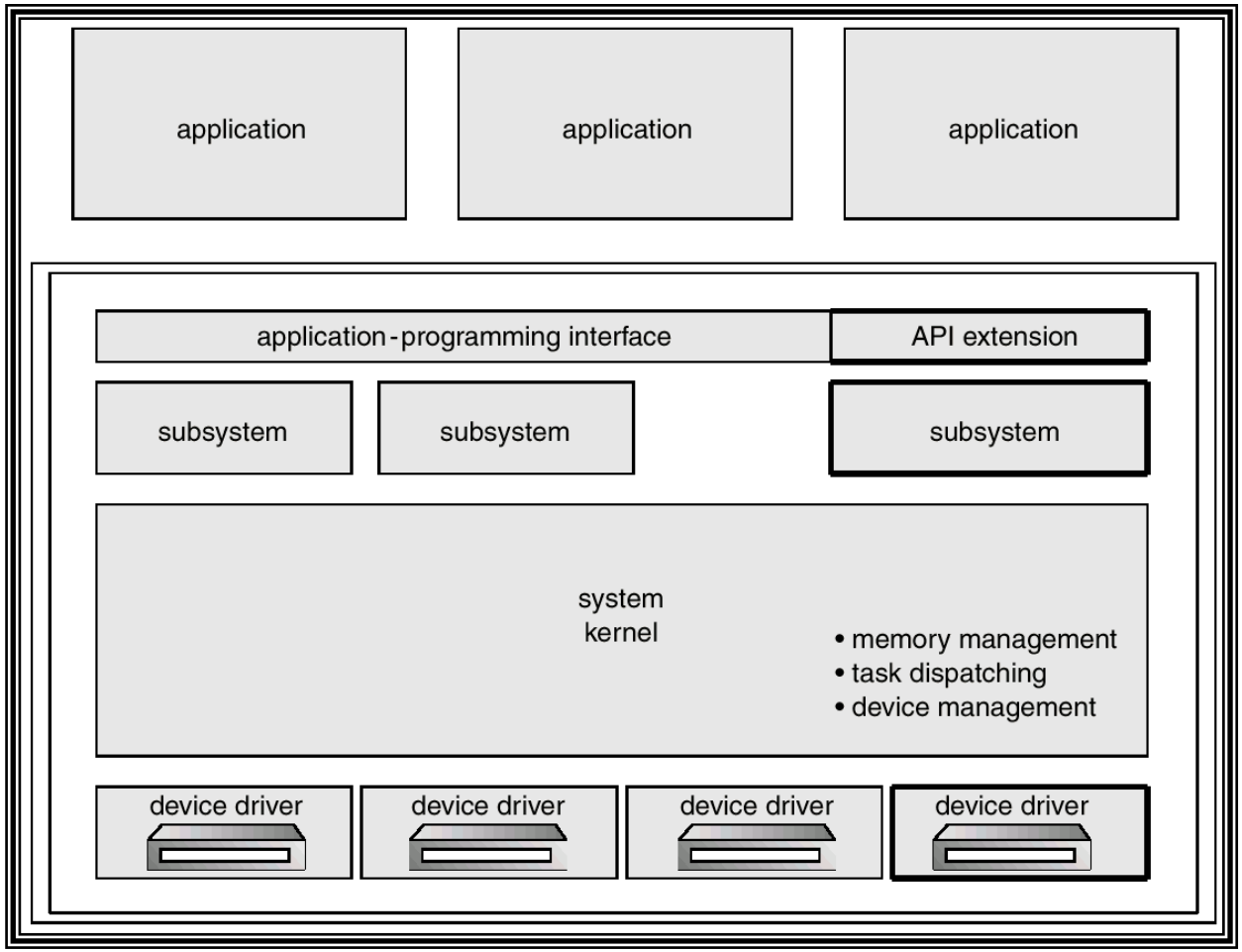
Layered Approach

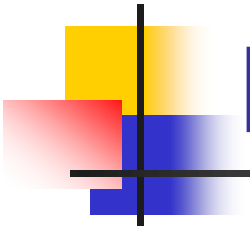
- The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level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

An Operating System Layer



OS/2 Layer Structure





Microkernel System Structure

- Moves as much from the kernel into "*user*" space.
- Communication takes place between user modules using message passing.
- Benefits:

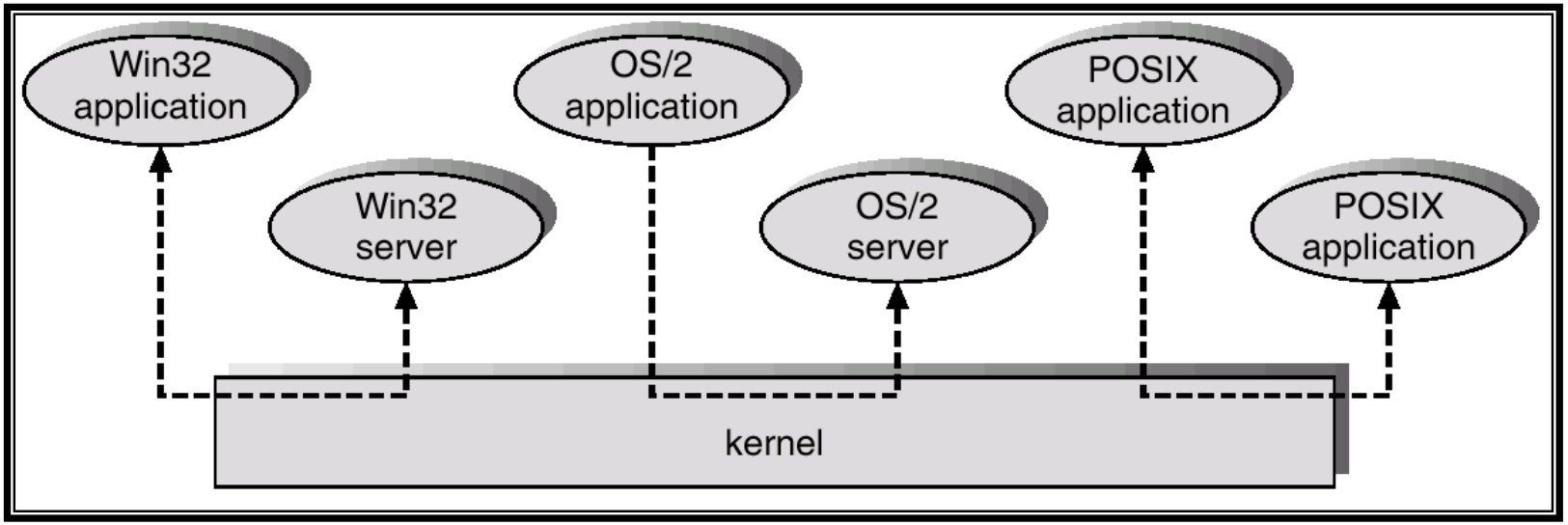
  - easier to extend a microkernel

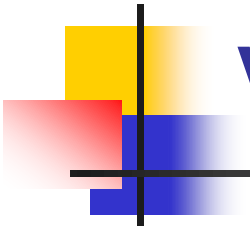
  - easier to port the operating system to new architectures

  - more reliable (less code is running in kernel mode)

  - more secure

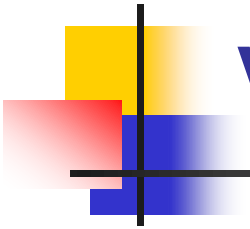
Windows NT Client-Server Structure





Virtual Machines

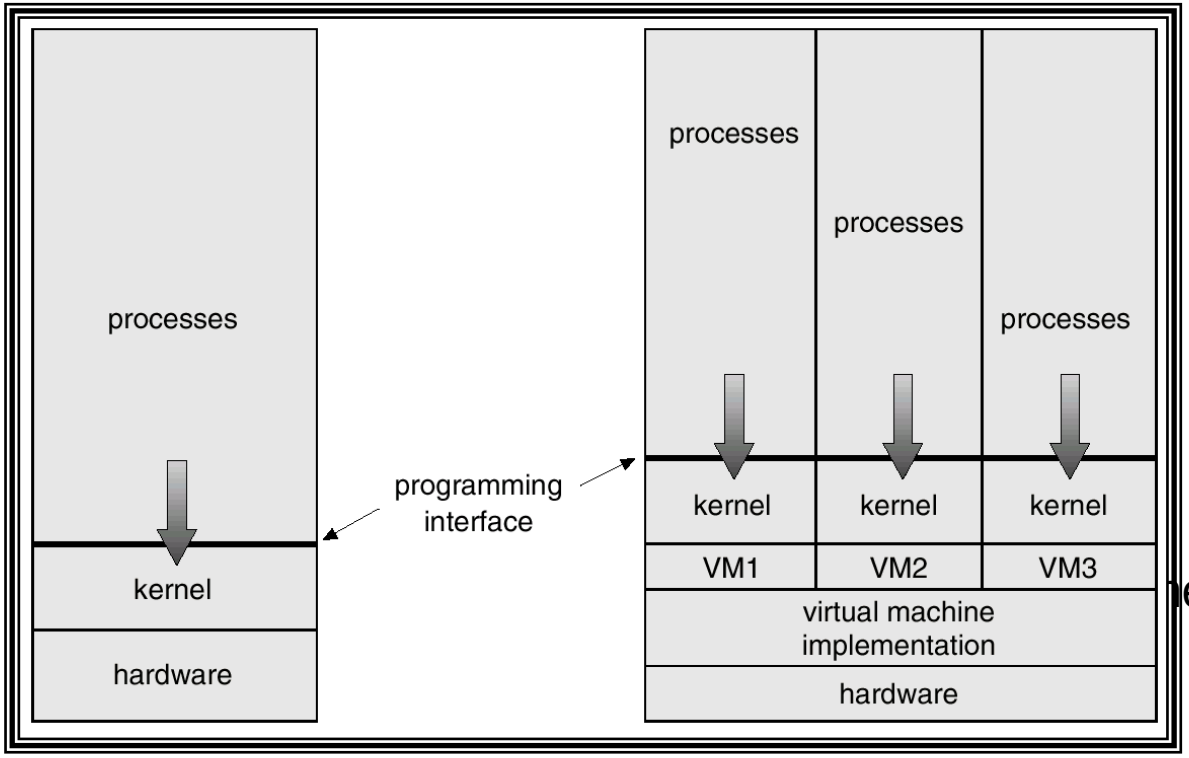
- A *virtual machine* takes the layered approach to its logical conclusion. It treats hardware and the operating system kernel as though they were all hardware.
- A virtual machine provides an interface *identical* to the underlying bare hardware.
- The operating system creates the illusion of multiple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



Virtual Machines (Cont.)

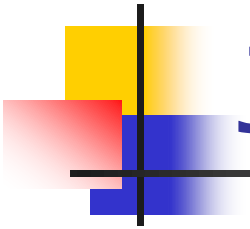
- The resources of the physical computer are shared to create the virtual machines.
  - CPU scheduling can create the appearance that users have their own processor.
  - Spooling and a file system can provide virtual card readers and virtual line printers.
  - A normal user time-sharing terminal serves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

System Models



Advantages/Disadvantages of Virtual Machines

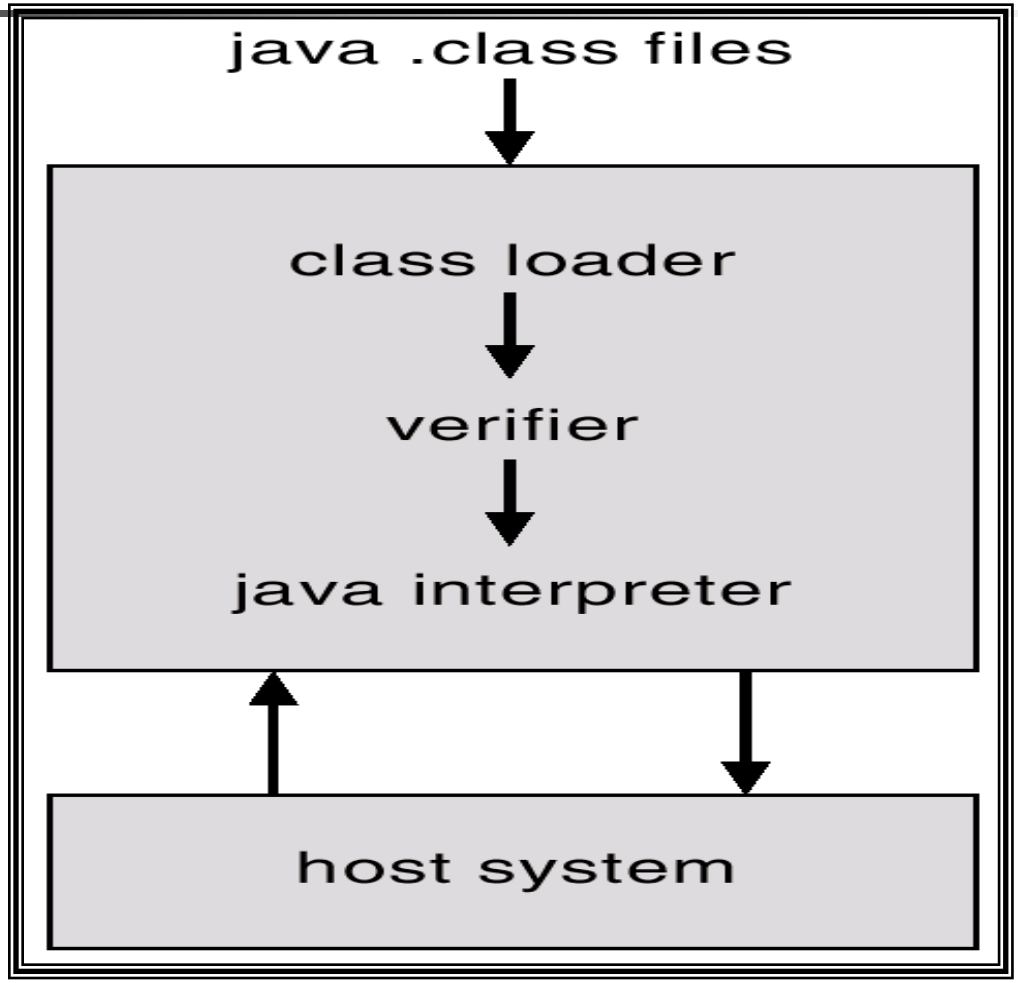
- The virtual-machine concept provides complete protection of system resources since each virtual machine is isolated from all other virtual machines. This isolation, however, permits no direct sharing of resources.
- A virtual-machine system is a perfect vehicle for operating-systems research and development. 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.

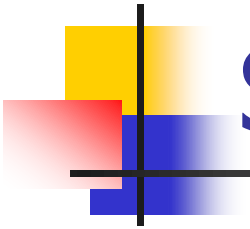


Java Virtual Machine

- 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

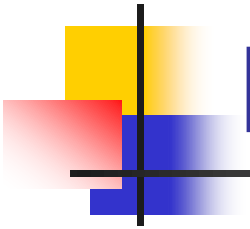
Java Virtual Machine





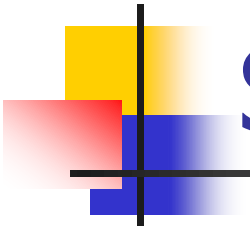
System Design Goals

- 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.



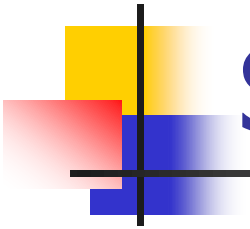
Mechanisms and Policies

- 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.



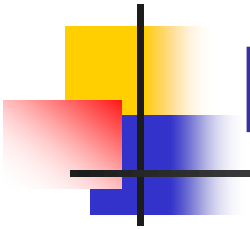
System Implementation

- 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.
  - 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.



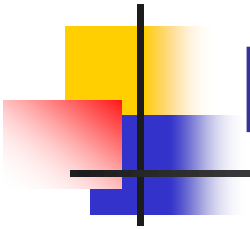
System Generation (SYSGEN)

- 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.



Lecture 1: Review Questions

- What are the two main functions of an operating system?
- What is the difference between kernel mode and user mode? Why is the difference important to an operating system?
- What is multiprogramming?
- What is spooling? Do you think that advanced personal computers will have spooling as a standard feature in the future?
- On early computers, every byte of data read or written was directly handled by the CPU (i.e., there was no DMA—Direct Memory Access). What implications does this organization have for multiprogramming?
- Why was timesharing not widespread on second-generation computers?



Lecture 1: Review Questions

- Which of the following instructions should be allowed only in kernel mode?
  - (a) Disable all interrupts.
  - (b) Read the time-of-day clock.
  - (c) Set the time-of-day clock.
  - (d) Change the memory map.
- List some differences between personal computer operating systems and mainframe operating systems.
- Give one reason why a closed-source proprietary operating system like Windows should have better quality than an open-source operating system like Linux. Now give one reason why an open-source operating system like Linux should have better quality than a closed-source proprietary operating system like Windows.