



Protection & Design Approaches

Date 20/01/2004

Hardware Protection

– Dual-Mode Operation

- Sharing system resources requires operating system to ensure that an incorrect program cannot cause other programs to execute incorrectly.
- 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 *supervisor mode* or *system mode*) – execution done on behalf of operating system.

- *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.



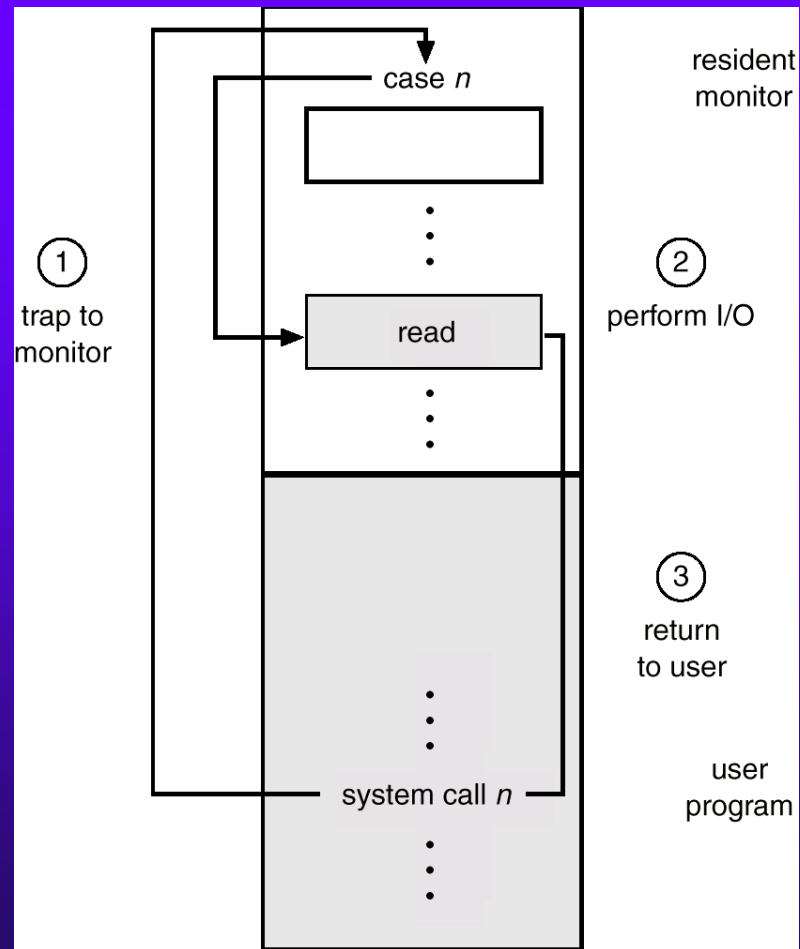
– I/O Protection

- 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).

– Memory Protection

- Must provide memory protection at least for the interrupt vector and the interrupt service routines.
- In order to have memory protection, add two registers that determine the range of legal addresses a program may access:
 - **base register** – holds the smallest legal physical memory address.
 - **Limit register** – contains the size of the range
- Memory outside the defined range is protected.

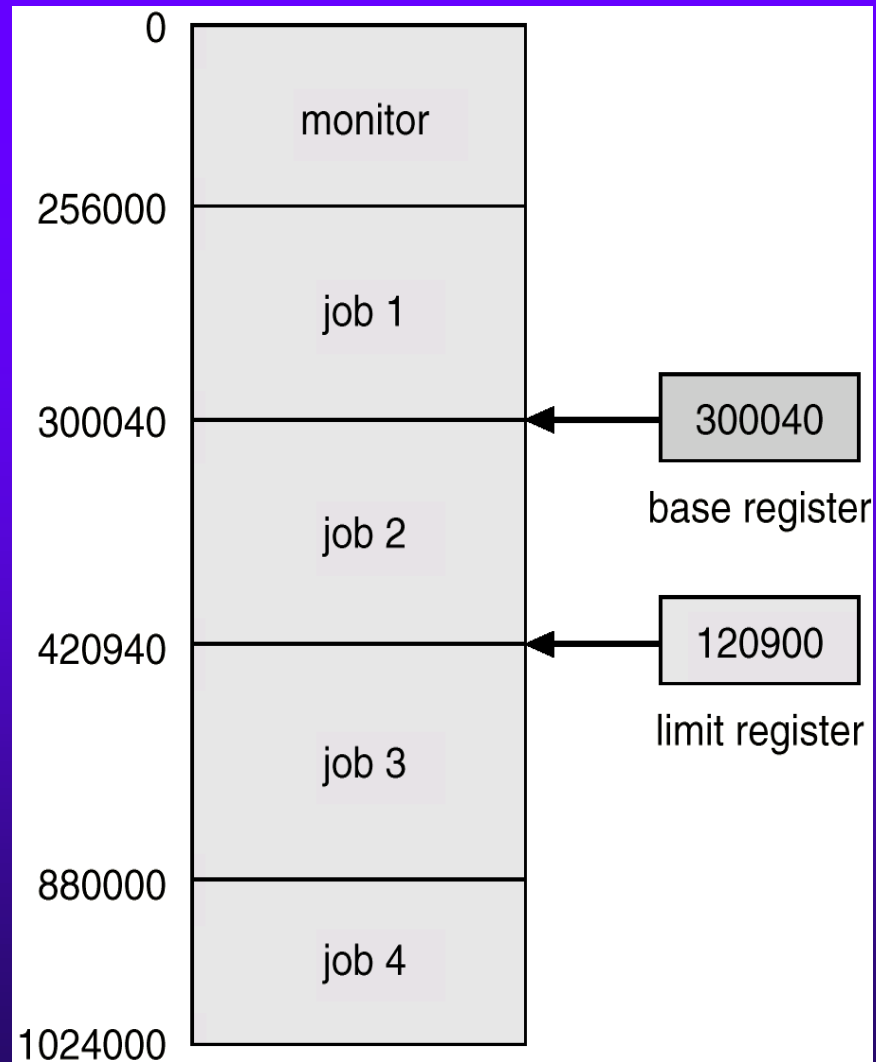
Use of A System Call to Perform I/O



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.

A Base And A limit Register Define A Logical Address Space





◆ CPU Protection

- *Timer* – interrupts computer after specified period to ensure 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.
- Time also used to compute the current time.
- Load-timer is a privileged instruction.

Operating-System Structures

- System Components
 - Process Management
 - A *process* is one instance of 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 and deadlock handling.
 - Process relationships (parent.child, process group ...)





- 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. It is the largest storage device CPU can address and access directly.
 - 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.
 - Maintains mapping from virtual to physical memory.



- Secondary-Storage Management

- Since main memory (*primary storage*) is volatile (persistent) 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

- I/O System Management

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



- 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.
- 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.



- Networking
 - 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.
 - A distributed system provides user access to various system resources.
 - Access to a shared resource allows:
 - Computation speed-up
 - Increased data availability
 - Enhanced reliability
- Command-Interpreter System
 - It is the interface between the user and the operating system.
 - Command interpreter can be either GUI or character interface.
 - 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



– 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 (for efficiency and protection), 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.

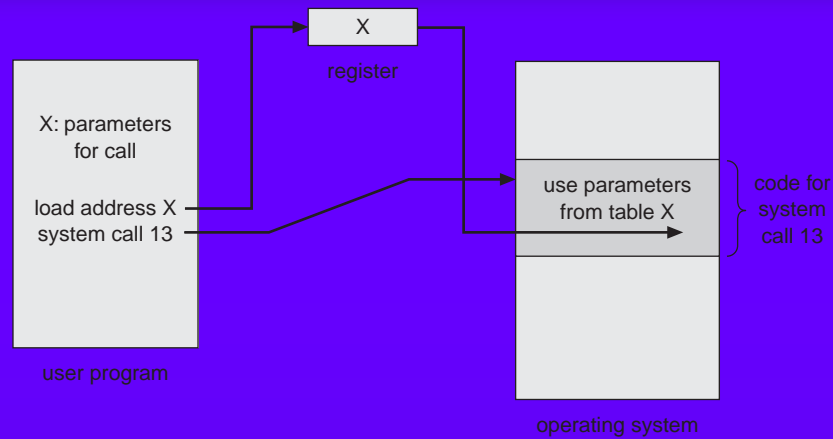
Additional functions exist not for helping the user, but rather for ensuring efficient system operations.

- Resource allocation – allocating resources(physical, logical) 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

- Discussion about copying one file's content to another file.
- 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. Bliss, PL/360)
- 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.



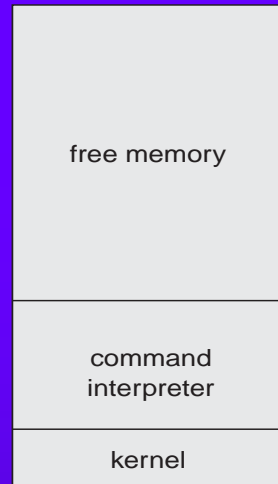
- Process control
 - End, Abort
 - Load, Execute
 - Create process, Terminate process
 - Get process attribute, Set process attribute
 - Wait for time
 - Wait event, Signal event
 - Allocate and free memory
- File management
 - Create file, Delete file
 - Open, Close
 - Read, Write, Reposition
 - Get and Set file attributes



- Device Management
 - Request and release device
 - Read, Write and Reposition
 - Get and Set device attributes
 - Logically attach or detach devices
- Information maintenance
 - Get time or date, set time or date
 - Get and Set system data
 - Get process, file, or device attributes
 - Set process, file, or device attributes
- Communication
 - Create, Delete communication connection
 - Send, Receive message
 - Transfer status information
 - Attach or detach remote devices



MS-DOS execution

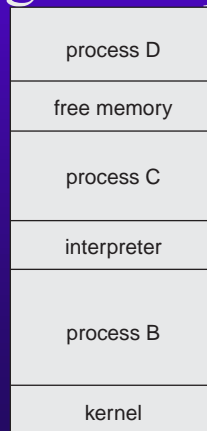


(a)



(b)

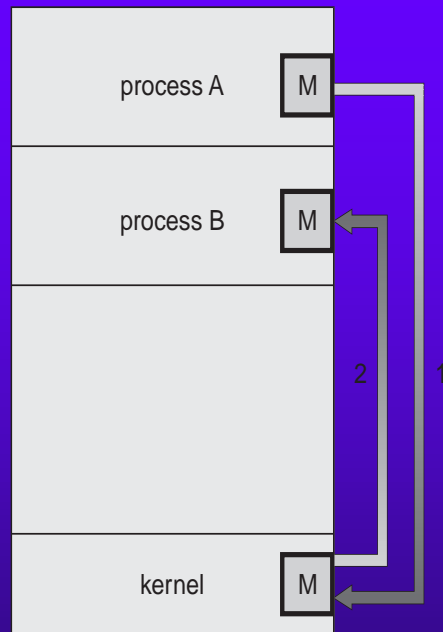
– Unix running multi programming



Communication models

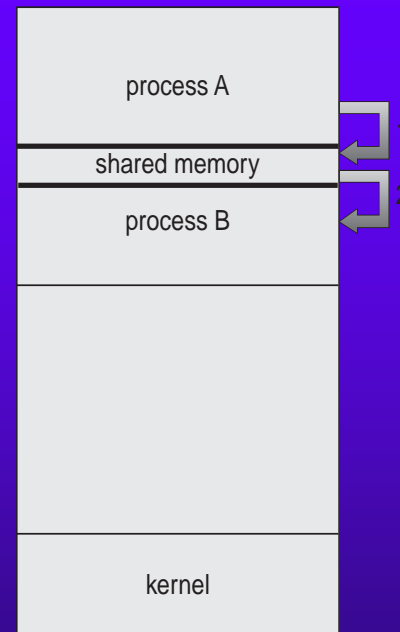


Message Passing



(a)

Shared memory



(b)



– 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.

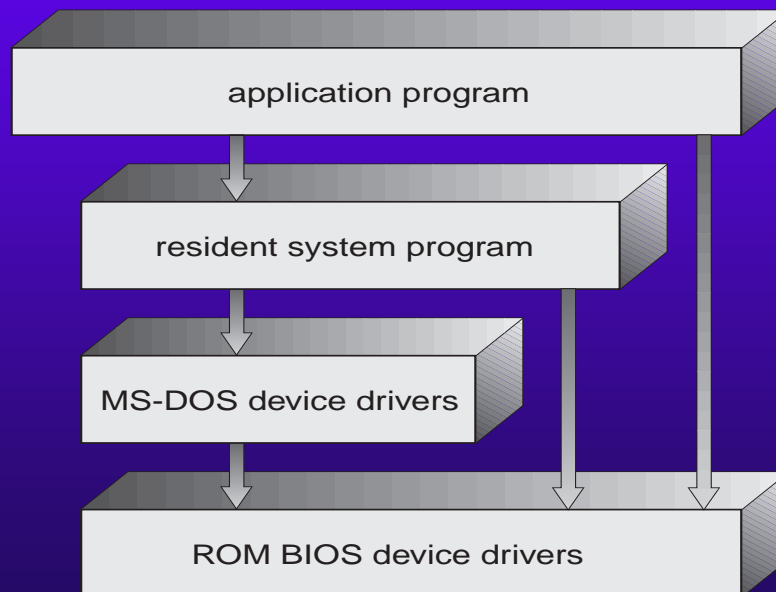
– System Structure

- Simple Approach

- 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





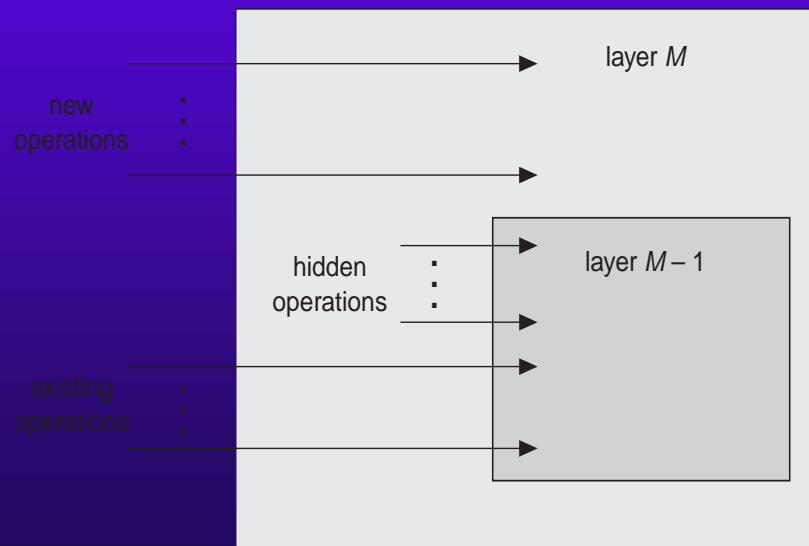
Unix System structure

(the users)		
shells and commands compilers and interpreters system libraries		
<i>system-call interface to the kernel</i>		
signals terminal handling character I/O system terminal drivers	file system swapping block I/O system disk and tape drivers	CPU scheduling page replacement demand paging virtual memory
<i>kernel interface to the hardware</i>		
terminal controllers terminals	device controllers disks and tapes	memory controllers physical memory



• 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.
- This approach simplifies debugging and system verifications.





- Difficulty of Layered approach
 - Careful definition of the layer is needed.
 - Less efficient than other types(Each layer adds overhead)
- Micro kernel approach
 - Kernel consist of only essential components
 - The kernel will be very small
 - Main function of the Micro kernel is to provide communication facility (using Message passing)
 - Advantages
 - Easy to extend, provide more security & reliability.
 - Examples QNX,MacOS X Server

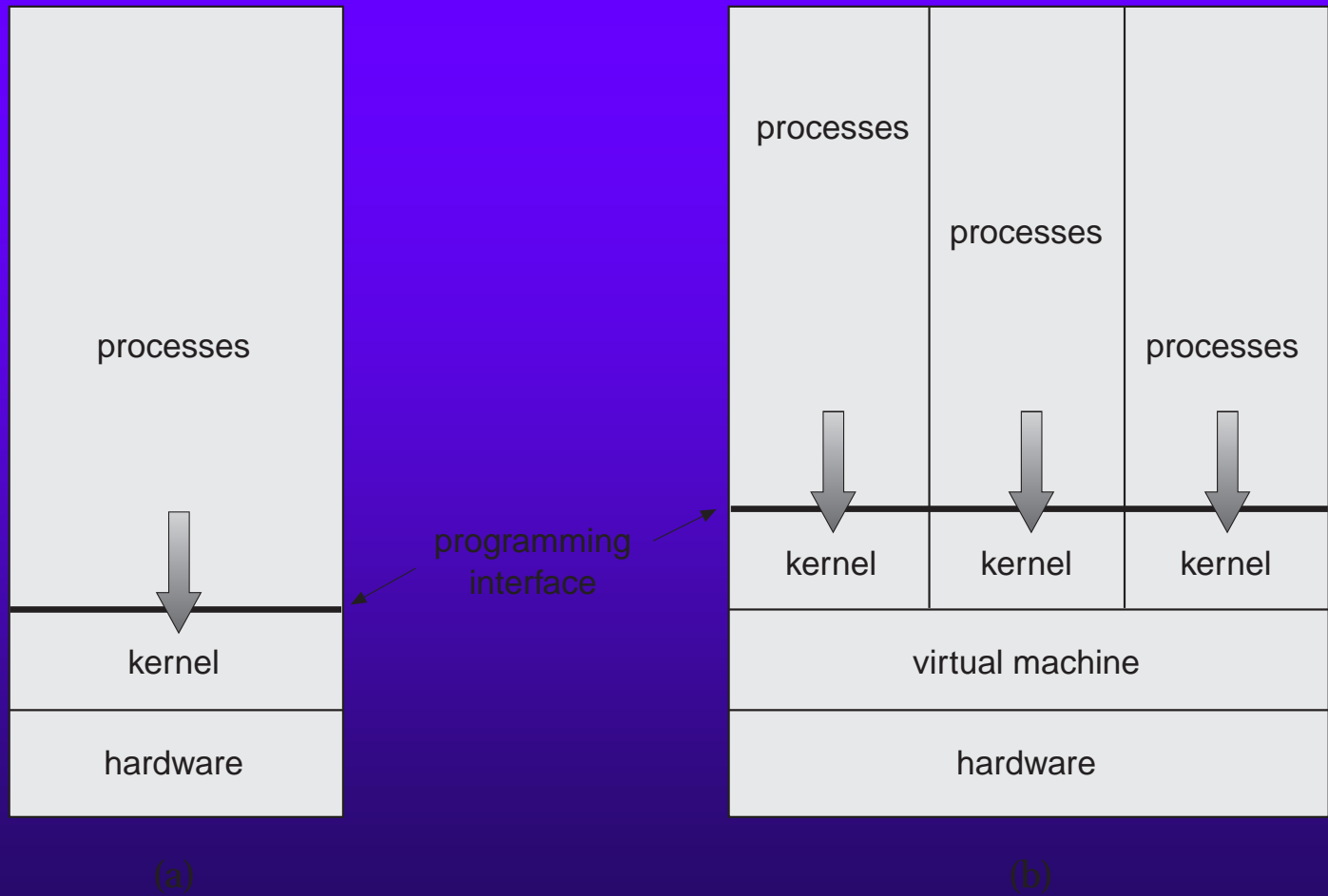


– Virtual Machines

- The operating system creates the illusion of multiple processes, each executing on its own processor with its own (virtual) memory.
- 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





- Advantage and disadvantage of VM implementation
 - 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.

– System Design and Implementation

• 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.