CAS CS 552
Intro to Operating Systems

Richard West

Operating System Structures

System Services

- Services required for:
  - Process management
  - Memory management
  - File management
  - I/O management
- Not all above management services are required for all OSes
  - e.g., embedded systems may not have any file system support & hence no need for file management
- Other management features?
  - Power management
  - Network resource management

Process Management

- What is a process?
  - A program in some state of execution
  - Includes the text + data areas of a program plus runtime state info including a process control block (in Linux this is a “task structure”), a stack and heap memory area
- Processes have their own address space i.e., range of memory addresses for code and data
  - Use resources, including the CPU to execute the instructions of the programs within their address spaces
  - Each process associated with at least one program counter value that keeps track of the next instruction to execute

OS Operations w.r.t. Process Management

- Creation/deletion of processes (can be user and system processes)
- Suspension/resumption of processes
- Process synchronization mechanisms
- IPC mechanisms
- Deadlock handling mechanisms (although not common on most systems)

Main memory Management

- CPU reads/writes main memory to load/store instructions and operands
- For systems with disk storage, programs are stored in files and loaded into main memory for execution
  - CPU cannot directly address secondary storage, only main memory

OS Responsibilities w.r.t. Memory Management

- Keep track of which parts of memory are currently being used and by whom
- Decide which processes to load into memory when space becomes available
- Allocate/de-allocate space as necessary
  - Can involve dynamic memory allocation during execution of a process
  - Can involve allocating space for newly-created processes
  - Can involve reclaiming memory from terminated processes and processes that no longer need dynamically-allocated memory

File/Storage Management

- A file is a logical storage unit holding a collection of related data
- OS responsible for:
  - Creation/deletion of files/directories
  - Primitives for manipulating files
    - Set of file operations for reading/writing/seeking…
    - Controlling access rights…
  - Mapping files onto secondary storage
    - Methods for lookup/indexing files
  - Free space management of secondary storage, storage allocation, disk scheduling algorithms…

I/O Management

- OS should hide peculiarities of hardware devices from the user
- I/O subsystem responsible for:
  - Memory management of I/O including buffering (storing data temporarily while it is being transferred), caching (storing parts of data in faster storage for performance)
  - General device-driver interface
  - Drivers for specific hardware devices

User Interaction

- For interactive systems, need a way for users to issue service requests
  - Command-line interface/shell that interprets text-based requests
  - Window-based i/fs (or GUIs) with icons, widgets etc

System Calls

- Need a method to pass control from user-level (or less privileged protection domain) to kernel-level, to request privileged services
- System calls, like function calls but control flow passes to more privileged kernel
- Syscalls, like function calls, often require arguments
  - e.g., read(fd, buffer, nbytes);

Syscall Parameter Passing

- 3 possible ways:
  - Registers
    - May be more parameters than registers
    - Arguments may be too large for registers
  - Via a stack
  - Via a block or table in memory whose address is passed as a parameter in a register

Syscall Categories

- Process control – creating, executing, exiting a process…
  - e.g., fork, exec, exit
- File manipulation
  - e.g., creat, open, close…
- Device manipulation
  - e.g., read, write, lseek…
- Information maintenance
  - e.g., getpid, stat…
- communications

System Programs

- Many system programs exist that interface to system calls
- System programs for:
  - File manipulation - e.g., Standard I/O library
  - Status information – user, time, process info…
  - Programming language support – compilers, assemblers, linkers, loaders
  - Communications – file transfer, remote login, telnet/ssh
- Application programs also exist that may be closely tied to a system
  - e.g., Web browsers, word processors, databases, text editors

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

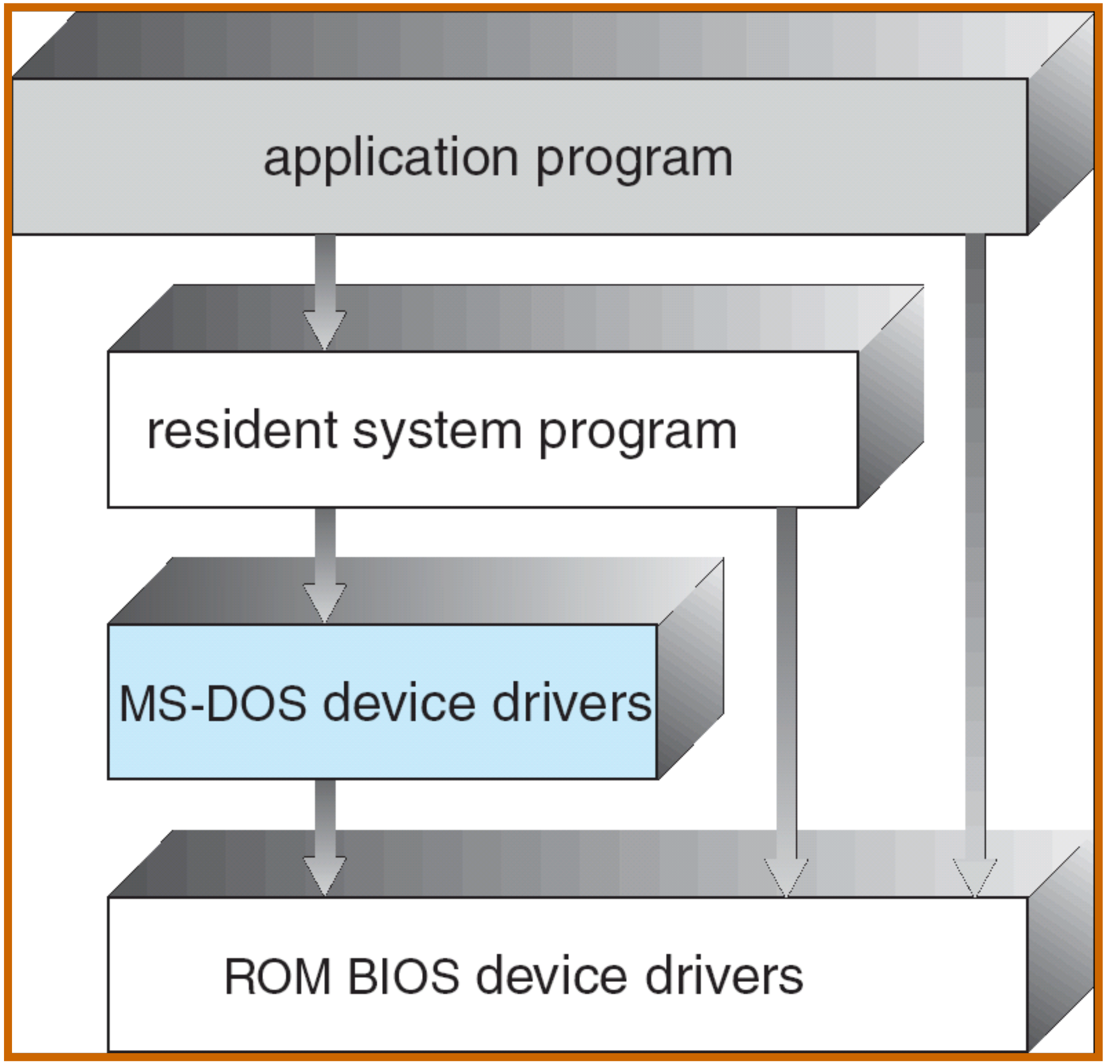
System Structure

- Monolithic – all services co-exist together with no clear separation of functionality e.g., MSDOS, Linux
  - Rather than having a monolithic system, divide it into smaller modular components with clean interfaces between various services
  - Easier to modify/extend – possibly more robust
- MSDOS has no clear separation of functionality – users can directly access basic I/O routines and manipulate hardware
- UNIX has evolved to be more layered but still largely monolithic
- Solaris, Linux and similar systems support kernel modules, making them extensible (e.g., supporting dynamically-loaded device drivers)
- Mach is an early instant of a micro-kernel

Simple Structure

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

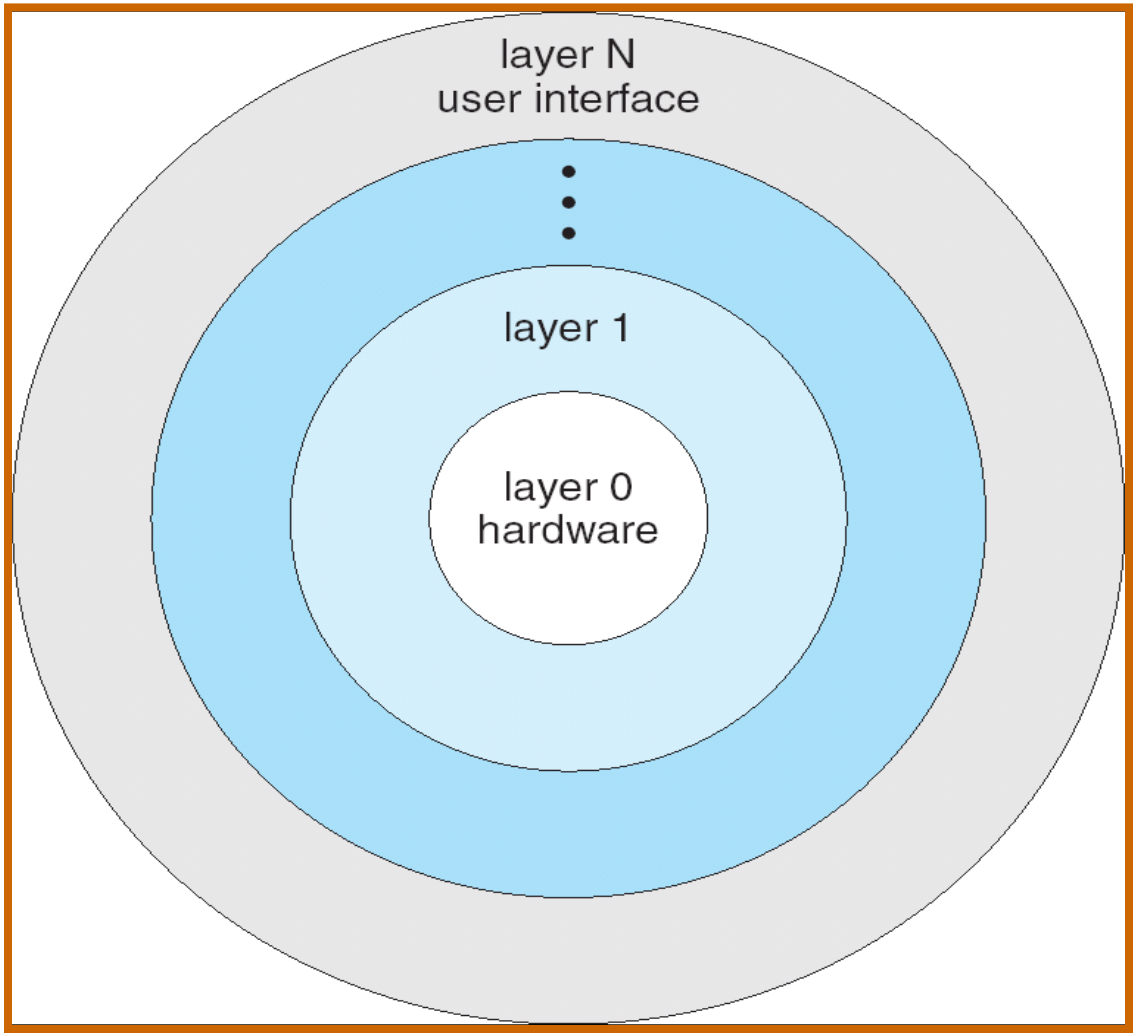
MS-DOS Layer Structure



Layered Approach

- System built in layers with each providing a specific service to layer immediately above
  - Functionality in different layers can be easily changed as long as interfaces between layers remain the same
  - Allows for easy debugging, development and isolation of problems when new functionality is added
- Each layer is built on service provided by lower layers
  - How services of lower layers are implemented is not necessary to know – only knowledge of what the services do is necessary
- Disadvantages?
  - A layer can only use services at a lower layer, so careful layer design is necessary
  - May induce additional overheads when services need to communicate via intermediate layers

Layered Operating System



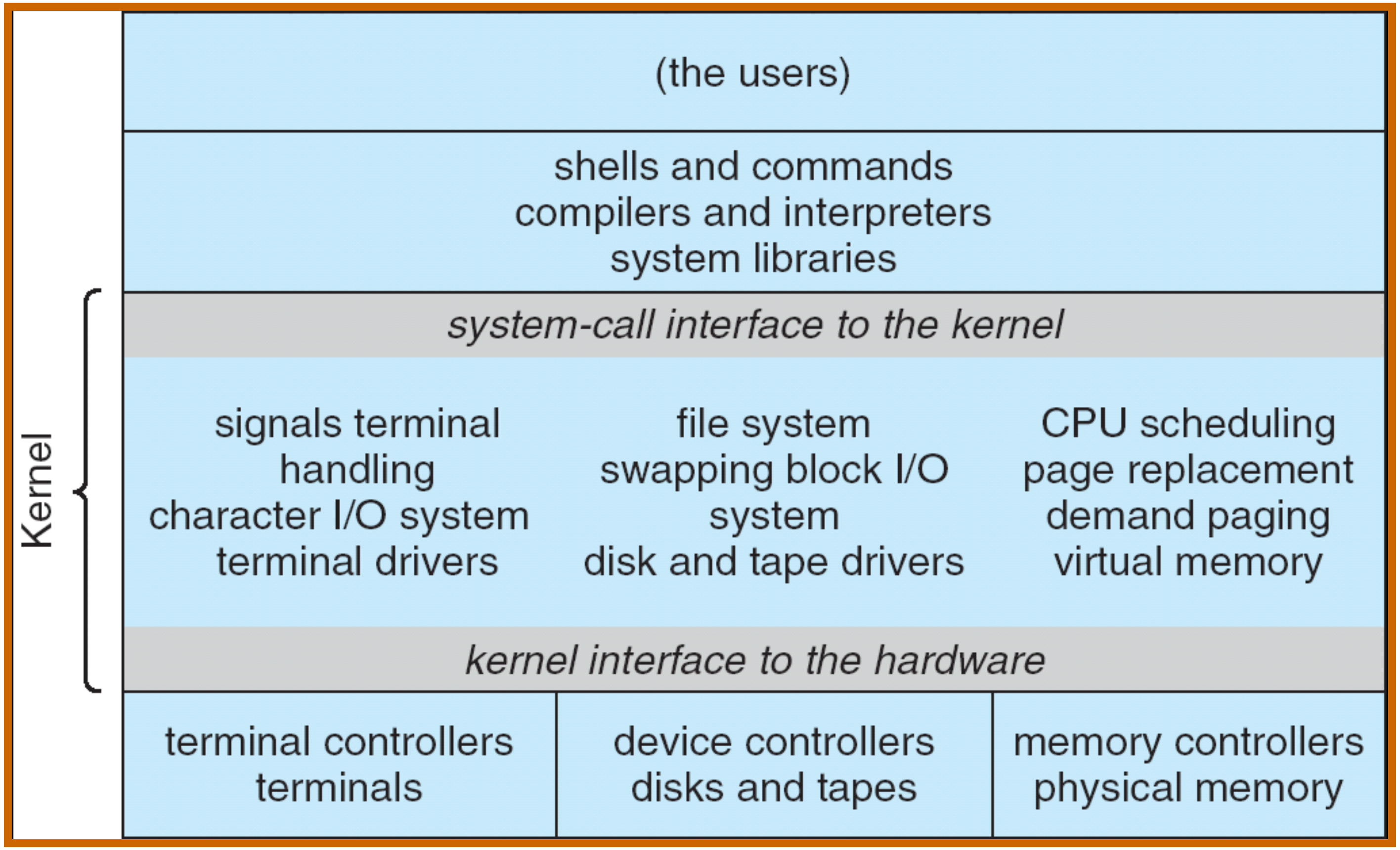
"THE"-Multiprogramming Layered System Example (Dijkstra)

- "THE" Layered System
  - Layer 4 – user programs
  - Layer 3 – I/O management
  - Layer 2 – message interpreter (operator/console device driver)
  - Layer 1 – segment controller/memory mgmt
  - Layer 0 – Processor allocation (CPU scheduling)

UNIX

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

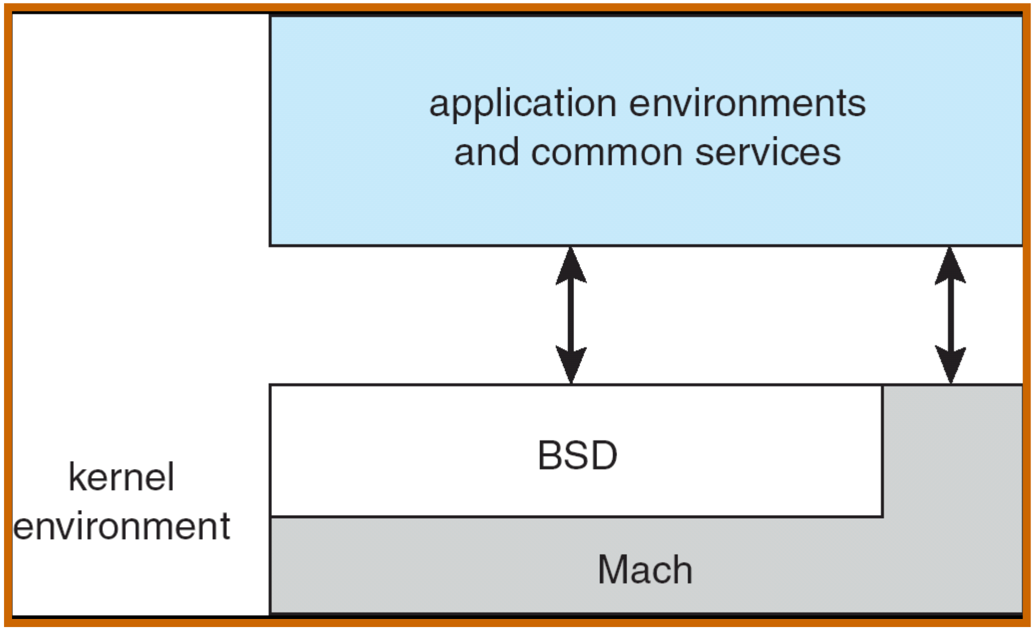
UNIX System Structure



Microkernel System Structure

- Moves all but basic abstractions 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
  - Reliability (less code is running in kernel mode)
  - Security
- Detriments:
  - Performance overhead of user space to kernel space communication

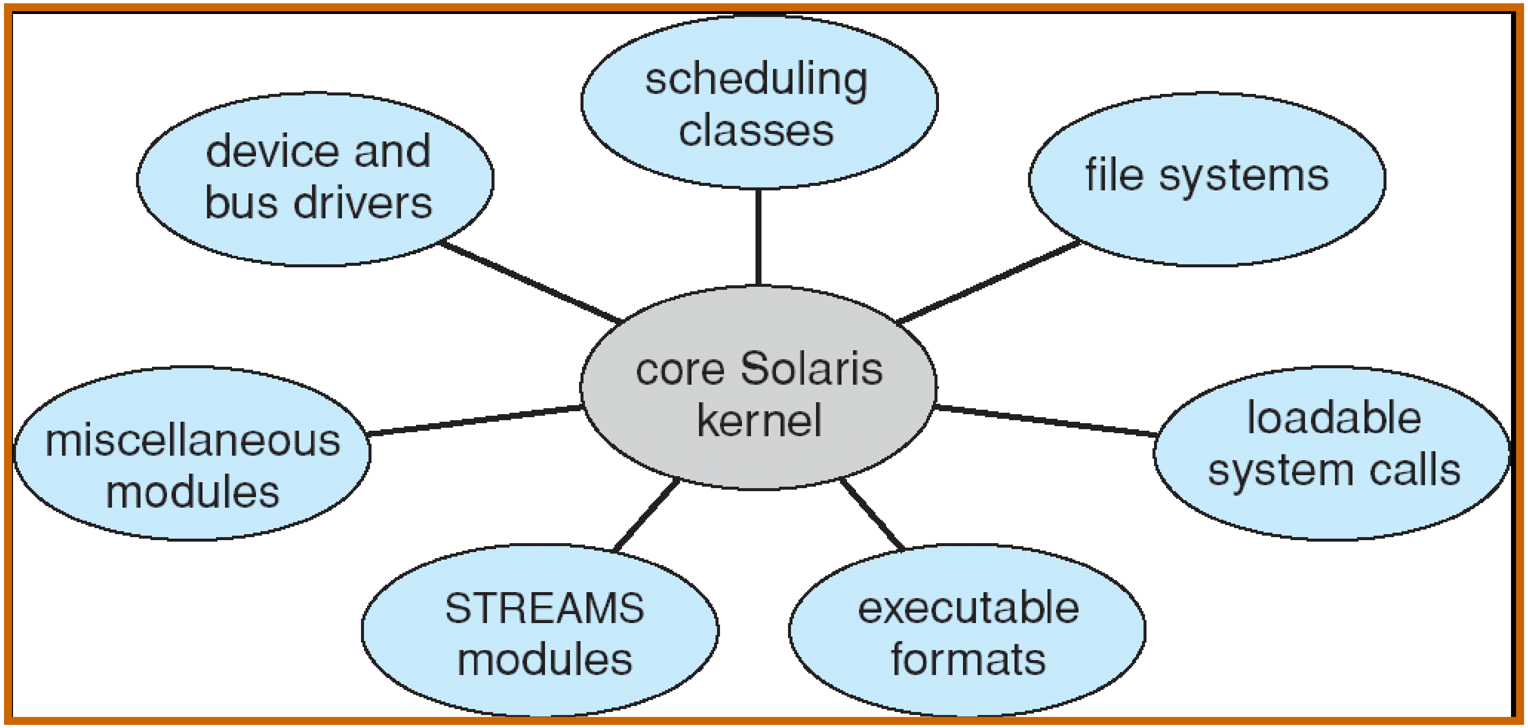
Mac OS X Structure



Modules

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

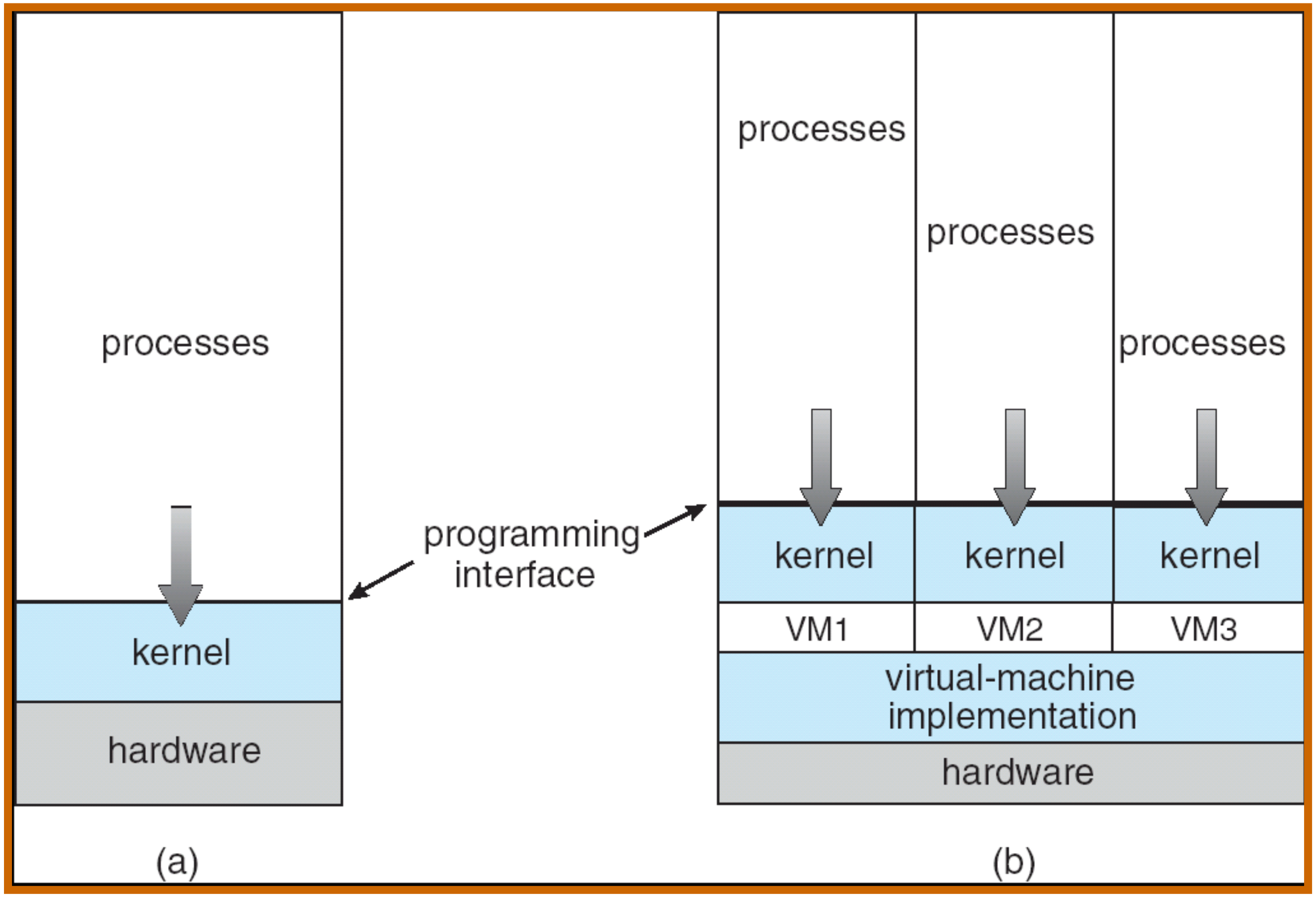
Solaris Modular Approach



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
- In its purest form, a virtual machine provides an interface *identical* to the underlying bare hardware
  - Paravirtualization involves modification to the guest(s) to run on physical platform
- A virtual machine creates the illusion of multiple processes, each executing on its own processor with its own (virtual) memory

Virtual Machines (Cont.)



(a) Nonvirtual machine (b) virtual machine

Virtual Machines (Cont.)

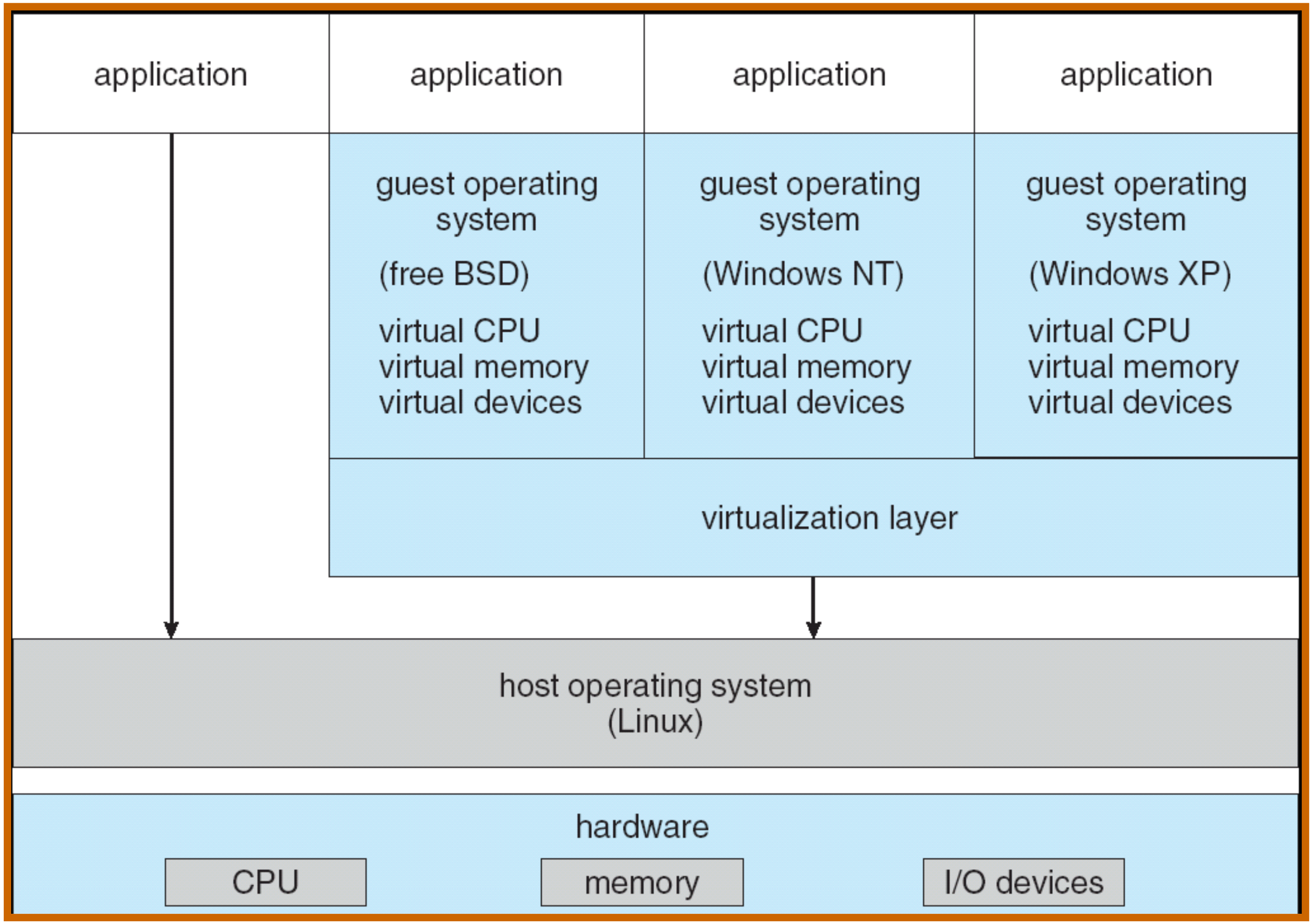
- The resources of the physical computer are shared to create the virtual machines
  - CPU scheduling can create the appearance that users have their own processor
  - Virtual disks can be created using files managed by an underlying file system

- Popek and Goldberg (1974) formal requirements:
  - **Equivalence** – to running directly on machine
  - **Control** – VMM in charge of virtualized resources
  - **Efficiency** – statistical fraction of machine instructions run without VMM intervention

Virtual Machines (Cont.)

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

VMware Workstation/Player Approach



The Java Virtual Machine