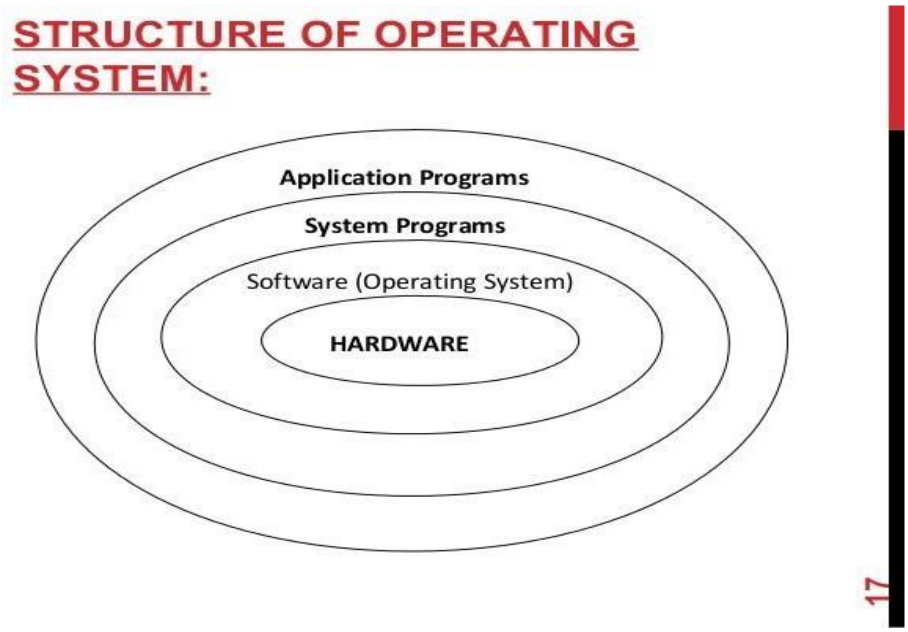


Operating System Design and Implementation

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

Important principle to separate

Policy: What will be done?

Mechanism: How to do it?

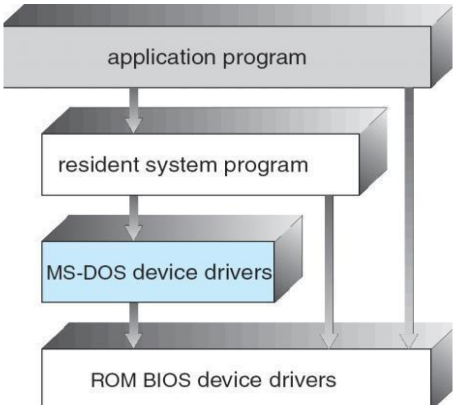
Mechanisms determine how to do something, policies decide what will be done

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

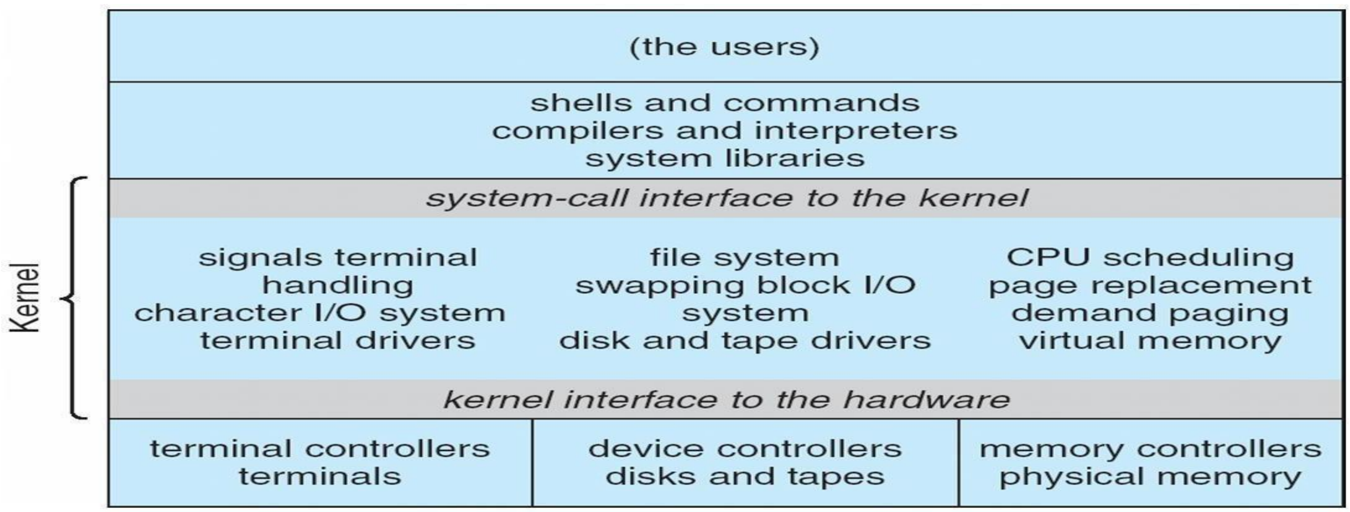
- MS-DOS – written to provide the most functionality in the least space Not divided into
- modules
- Although MS-DOS has some structure, its interfaces and levels of Functionality are not well separated

MS-DOS Layer Structure



- 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

Traditional UNIX System Structure



UNIX

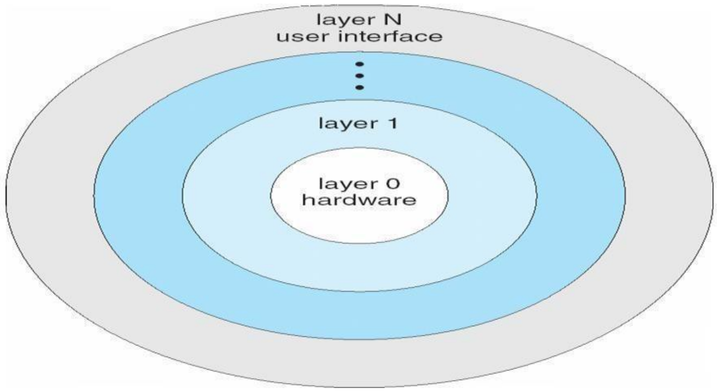
- UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring. The UNIX OS consists of two separable parts

- 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

Layered Operating System



Micro kernel 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

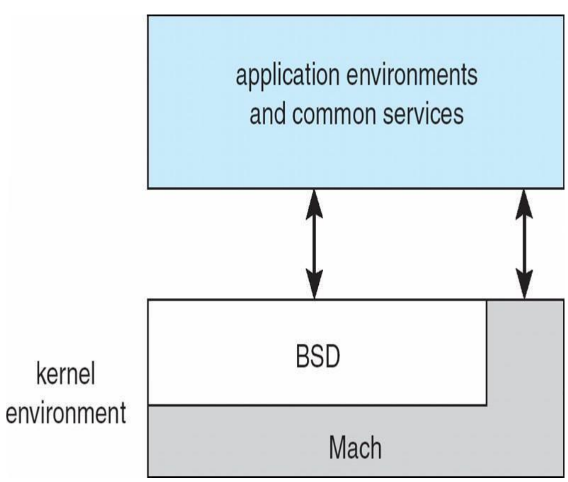
Easier to port the operating system to new architectures More reliable (less code is running in kernel mode)

More secure

Detriments:

Performance overhead of user space to kernel space communication

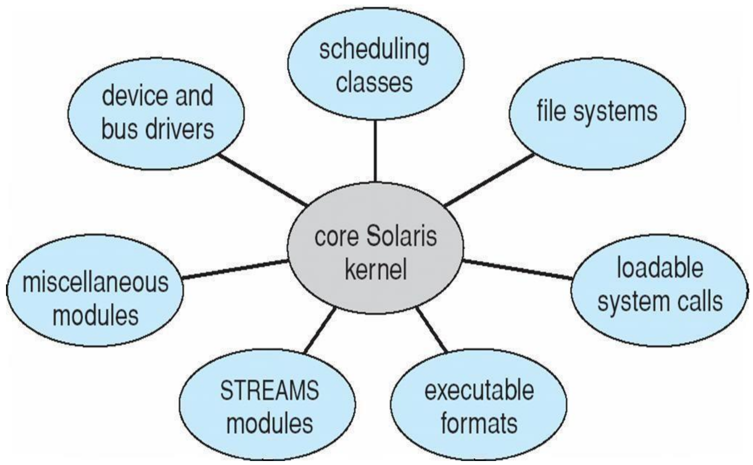
MacOS X Structure



Modules

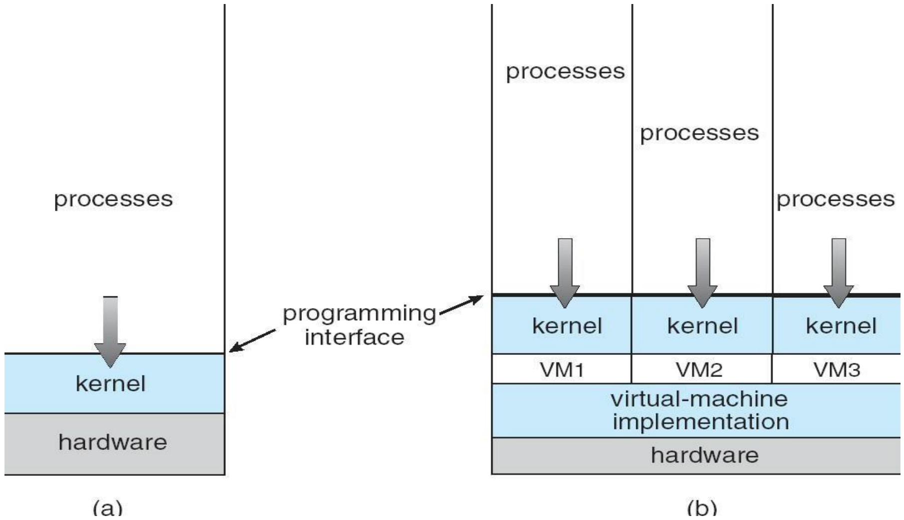
Most modern operating systems implement kernel modules
Uses object-oriented approach
Each core component is separate
Each talks to the others over known interfaces Each is loadable as needed within the kernel Overall, similar to layers but with more flexible

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
Commutate 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



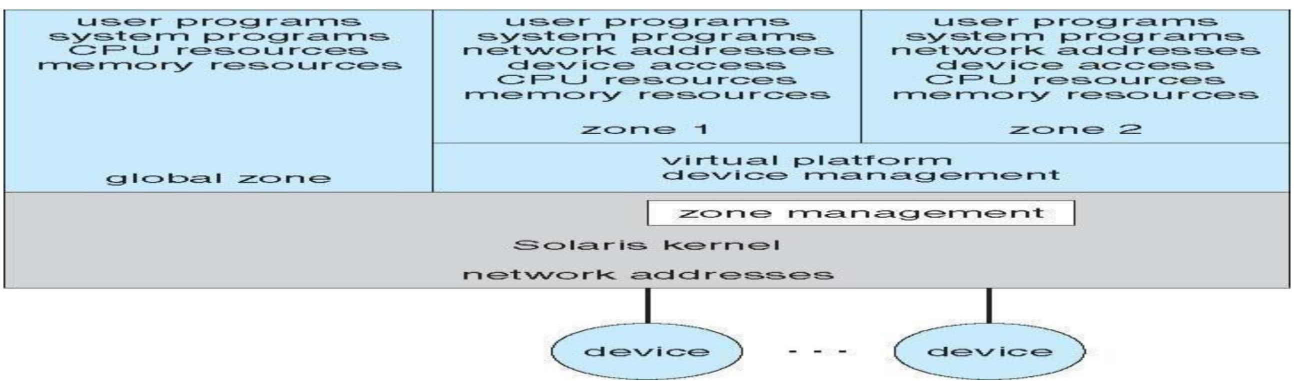
Para-virtualization

Presents guest with system similar but not identical to hardware

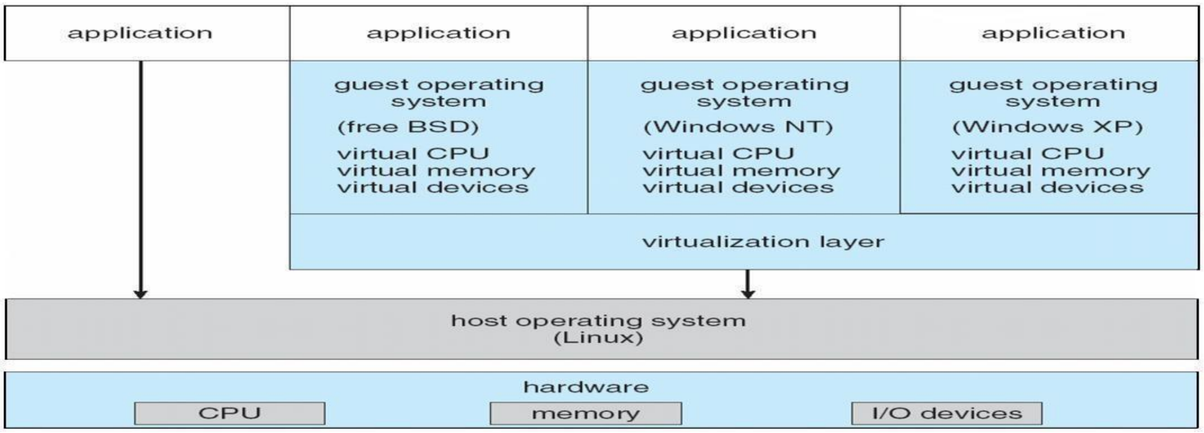
Guest must be modified to run on par virtualized hardware

Guest can be an OS, or in the case of Solaris 10 applications running in containers

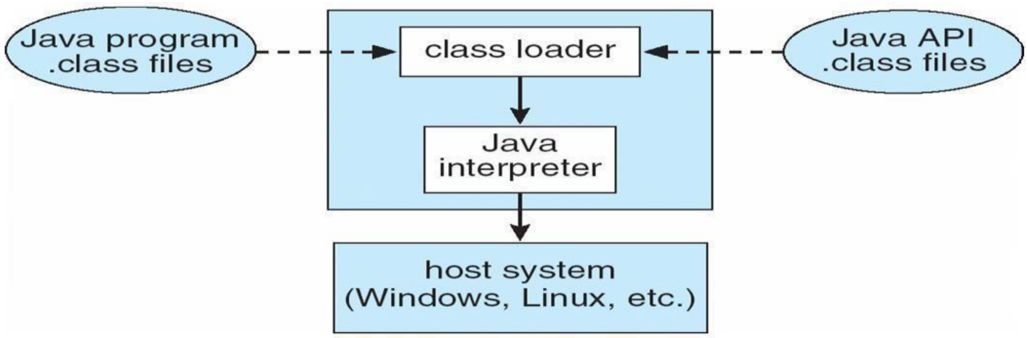
Solaris 10 with Two Containers



VMware Architecture



The Java Virtual Machine



Operating-System Debugging

Debugging is finding and fixing errors, or bugs generate log files containing error information

Failure of an application can generate core dump file capturing memory of the process Operating system failure can generate crash dump file containing kernel memory Beyond

crashes, performance tuning can optimize system performance
Kernighan’s Law: “Debugging is twice as hard as writing the code in the rst place. Therefore, if youwrite the code as cleverly as possible, you are, by definition, not smart enough to debug it.”
DTrace tool in Solaris, FreeBSD, Mac OS X allows live instrumentation on production systems Probes fire when code is executed, capturing state data and sending it to consumers of those probes