TDIU11
Operating systems

Operating System Structures and Virtual Machines

[SGG7/8] Chapter 2.7-2.8
[SGG9] Chapter 2.7, 1.11.6

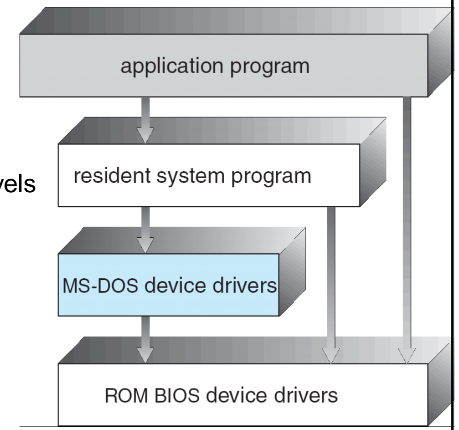
Christoph Kessler, IDA,
Linköpings universitet.

Operating System Structures

- How to manage OS complexity?
  - Divide-and-conquer!
  - Decompose into smaller components with well-defined interfaces and dependences
    - Layered Approach
    - Microkernels
    - Modules
    - Virtual Machines

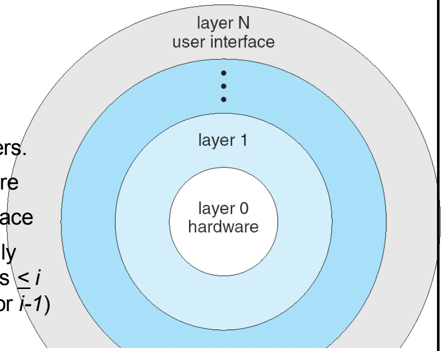
Simple Structure

- 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

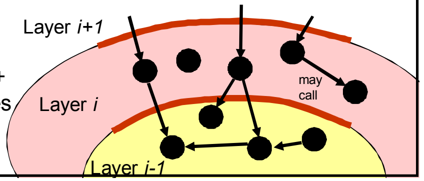


Layered Approach

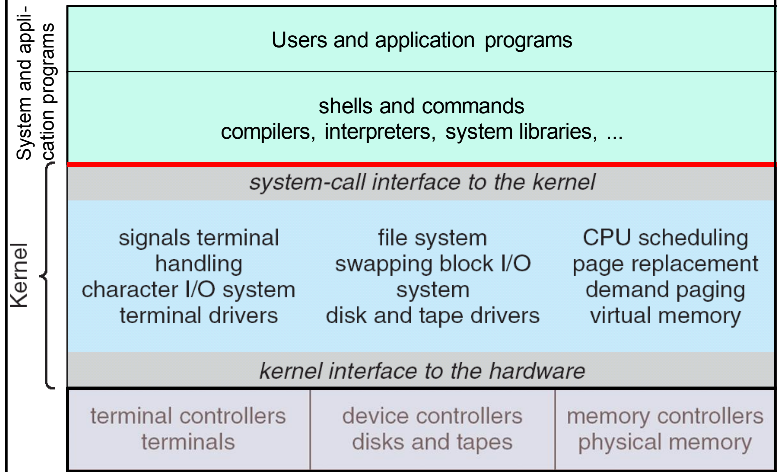
- The operating system is divided into a number of **layers** (levels, rings), each built on top of lower layers.
  - Bottom layer (0) = hardware
  - Top layer (*N*) = user interface
  - Functions in layer *i* call only functions/services in layers < *i* (**strict layering:** only in *i* or *i-1*)



- **Modularity**
  - **Interface** of a layer: upwards-exposed services + downwards-required services

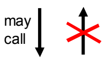


UNIX System Structure: 3 Layers



THE OS: 6 Layers

- A layered design was first used in the THE operating system
  [Dijkstra'68, Technische Hogeschool at Eindhoven, NL]



layer 5:	user programs
layer 4:	buffering for input and output
layer 3:	operator-console device driver
layer 2:	memory management
layer 1:	CPU scheduling
layer 0:	hardware

see SGG7 Ch. 23.4, p. 847

Problems of the layered approach

- **Cyclic dependences** between different OS components

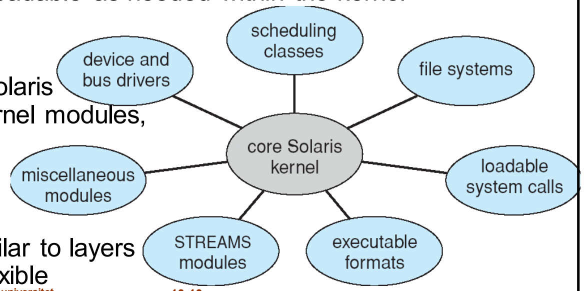
  Example:
  - Backing store driver for swapping should be able to call CPU scheduler to release the CPU while waiting for I/O
  - CPU scheduler needs to know about memory needs of all active processes, on a large system this information resides in memory that is possibly swapped out...
- **Less efficient**
  - Long call chains (e.g. I/O) down to system calls, possibly with parameter copying/modification at several levels
- Compromise solution: Have few layers → less structured

Microkernel System Structure

- "Lean kernel": Moves as much service functionality as possible from the kernel into "*user*" space
  - Kernel: Minimal process and memory management; IPC
- Communication between user modules by message passing
- Example: Mach kernel, used e.g. in Tru64 Unix or Mac OS-X
- 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

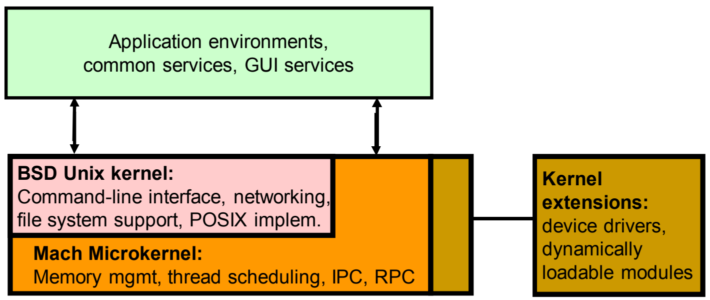
- Most modern operating systems implement kernel modules
- Component-based approach:
  - Each core component is separate
  - Each talks to the others over known interfaces
  - Each is loadable as needed within the kernel
- Example: Solaris loadable kernel modules, Linux, Mac-OS X



- Overall, similar to layers but more flexible

Example: Mac-OS X "Darwin"

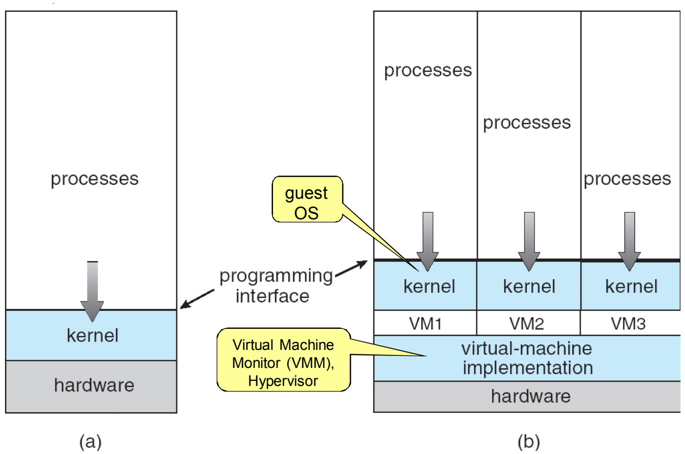
- Hybrid structure: Layering + Microkernel + Modules



Virtual Machines

- A ***virtual machine*** provides an interface *identical* to the underlying bare hardware (or to some other real or fictive machine).
  - Example: Multitasking OS creates the illusion that each process executes on its own (virtual) processor with its own (virtual) memory.
  - Example: qemu (used in Pintos labs) simulates x86 hardware
  - Example: The Java VM simulates an abstract computer that executes Java bytecode.
- Virtual machine implementation (**VM monitor**, **hypervisor**) intercepts operations and interprets them.
- Several virtual machines may share the resources of a physical computer:
  - CPU scheduling: create illusion that users have their own processor
  - Virtual disks with virtual file systems on physical disk / file system
  - A normal user time-sharing terminal serves as the virtual machine operator's console
- Can run multiple and different OS's on the same physical computer
  - Examples: VMware, Xen

Virtual Machines (Cont.)

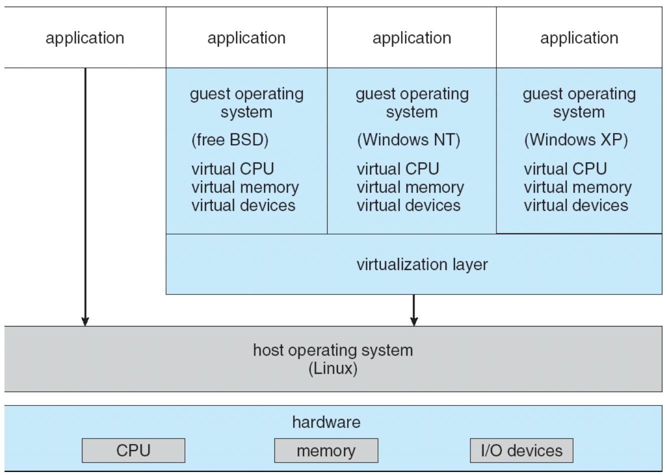


(a) Nonvirtual machine (b) Virtual machines

Virtual Machines – Advantages, Drawbacks

- Complete **protection** of system resources since each virtual machine is isolated from all other virtual machines.
  - however, permits no direct sharing of resources.
- Perfect vehicle for operating-systems **research, development, teaching**
  - System development is done on the virtual machine, instead of on a physical machine and so does not disrupt normal system operation.
- **Portability** across multiple platforms (host OS, hardware)
  - Java VM
  - Legacy binary codes for obsolete hardware still operational
- Saves appl.-**server hardware costs** if clients demand a private system
  - "Most servers today run at <15% utilization, TCO ~10k$/yr/server" [Xen]
- Difficult to implement
  - to provide an *exact* duplicate to the underlying machine
- Old idea!
  - 1972 by IBM on System/360

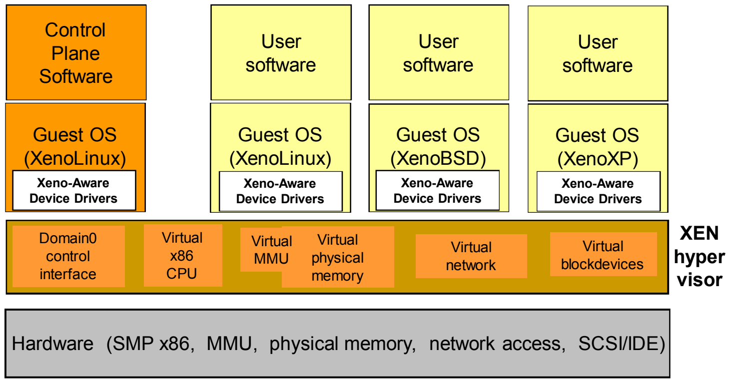
VMware Architecture



Virtualization technology overview

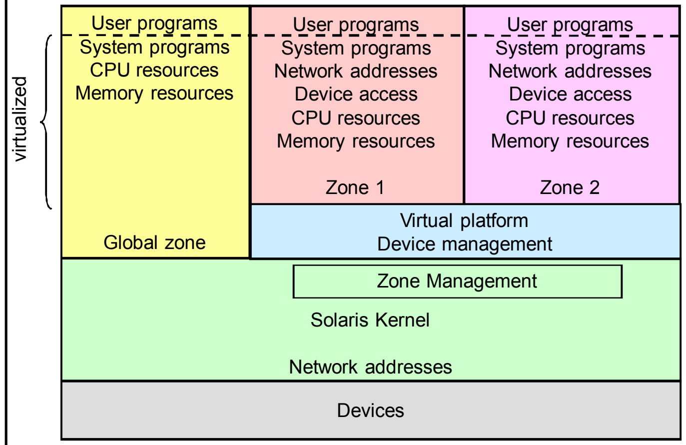
Traditional virtualization: **Virt. machine/Emulation**	Lightweight virtualization: **Paravirtualization**	Lightweight virtualization: **OS-level virtualization**
Emulates real or fictitious hardware + guest HW != host HW possible + diff. guest OSs possible + guest OS is not aware of the host OS beneath - VMM needed to dis-patch virtual kernel mode privileged instructions - translation overhead	"Almost" same hardware (barring speed and size) + different host and guest OSs possible - VMM needed - guest OS to be rewritten to be VMM-aware (fix privileged instructions) - still overhead → # of VMs limited	Virtualization done by the host OS No VMM Guest OS = Host OS (and same HW) + Multiple instances of same OS possible + Low overhead + Can scale up to hundreds of VMs e.g. for virtual private servers
VM/370 (IBM), VMware, Bochs, QEMU, Parallels, Microsoft Virtual Server, Java JVM, C#/.NET CLR	Xen UML (User-mode Linux) Denali	Docker for Linux Solaris 10 "Zones" (Containers), OpenVZ, Virtuozzo, Linux-VServer, FreeBSD-Jails

Paravirtualization Example: Xen

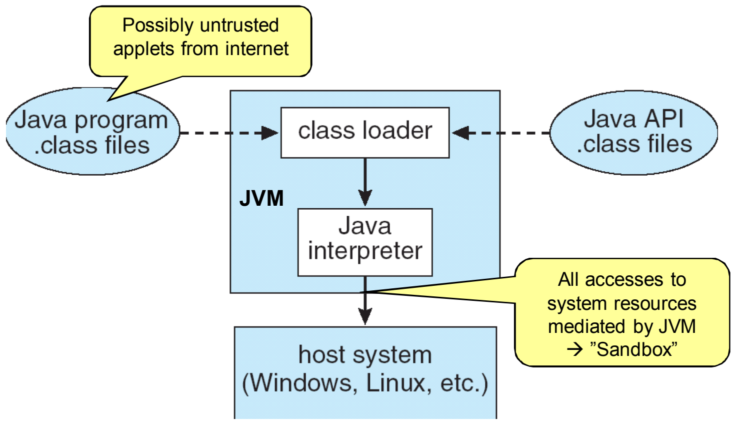


Adapted from: P. Barham et al.: Xen and the Art of Virtualization. Proc. SOSP 2003

OS-Level Virtualization Example: Solaris 10 Containers (”Zones”)



The Java Virtual Machine



Hardware support for virtualization

- Simulate mode bit, system call effects, … in software???
- Second hardware mode bit in status register
  - *Physical mode bit* used only by VMM / host kernel
    - Virtual machine incl. guest OS runs in physical user mode
  - *Virtual mode bit* used by guest OS
    - Virtual kernel mode vs virtual user mode
- AMD: *host mode* vs *guest mode*
  - Virtual machines run in guest mode
    - Completely unaware of the virtualization
    - Access to virtualized devices traps to VMM/host OS
  - VMM / virtualizing kernel can switch to host mode

Summary: Operating System Structures

- How to manage OS complexity?
  - Divide-and-conquer!
  - Decompose into smaller components with well-defined interfaces and dependences
    - Layered Approach
    - Microkernels
    - Modules
    - Virtual Machines
      - Traditional Virtualization
      - Light-Weight Virtualization (Paravirtualization, OS-level virtualization)

Literature: Virtual Machines, Virtualization

- ***IEEE Computer*** May 2005 special issue on Virtual Machines e.g.
  R. Uhlig *et al*.: Intel Virtualization Technology.
  IEEE Computer, May 2005, pp. 48-56.
- XenSource: *XenTM: Enterprise Grade Open Source Virtualization: Inside XenTM 3.0*. White Paper V06012006, www.xensource.com
- P. Barham *et al*.: Xen and the Art of Virtualization.
  Proc. of SOSP 2003, pp. 164-177, ACM press.
- S. Bellovin: Virtual Machines, Virtual Security?
  Communications of the ACM **49**(10): 104, Oct. 2006
- M. Price: The Paradox of Security in Virtual Environments.
  IEEE Computer Nov. 2008, pp. 22-28.
- And many others...