Operating System Structures and Virtual Machines

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

Copyright Notice: The lecture notes are mainly based on Silberschatz’s, Galvin’s and Gagne’s book (“Operating System Concepts”, 7th ed., Wiley, 2009). No part of the lecture notes may be reproduced in any form, due to the copyrights reserved by Wiley. These lecture notes should only be used for internal teaching purposes at the Linköping University.

Christoph Kessler, IDA, Linköpings universitet.

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 users)
- shells and commands
- compilers and interpreters
- system libraries

Kernel

- 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

Kernel interface to the hardware

- terminal controllers
- terminals
- device controllers
- disks and tapes
- memory controllers
- physical memory

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

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.)
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 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 across multiple platforms (host OS, hardware)
- “Most servers today run at <15% utilization, TCO ~10k$/yr/server” [Xen]
- Old idea! Difficult to implement.
- Saves appl.-dev. efforts, but # of VMs limited
  - Java VM
  - Portability

Legacy binary codes for obsolete hardware still operational across multiple platforms (host OS, hardware)

“Most servers today run at <15% utilization, TCO ~10k$/yr/server” [Xen]

Old idea! Difficult to implement.

Saves appl.-dev. efforts, but # of VMs limited
  - Java VM
  - Portability

Virtualization technology overview

<table>
<thead>
<tr>
<th>Traditional virtualization</th>
<th>Paravirtualization</th>
<th>Hardware virtualization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Emulates real or fictitious hardware</td>
<td>Virtually emulates hardware</td>
<td>Direct mapping of virtual resources to hardware</td>
</tr>
<tr>
<td>guest HW = host HW possible</td>
<td>guest OS = host OS possible</td>
<td>guest HW = host HW</td>
</tr>
<tr>
<td>guest OS is not aware of the host OS beneath</td>
<td>guest OS is not aware of the host OS beneath</td>
<td>guest OS is not aware of the host OS beneath</td>
</tr>
<tr>
<td>VMM needed to dispatch virtual kernel</td>
<td>VMM needed to dispatch virtual kernel</td>
<td>VMM needed to dispatch virtual kernel</td>
</tr>
<tr>
<td>mode privileged instructions</td>
<td>mode privileged instructions</td>
<td>mode privileged instructions</td>
</tr>
<tr>
<td>- translation overhead</td>
<td>- translation overhead</td>
<td>- translation overhead</td>
</tr>
<tr>
<td>VM/370 (IBM), VMware, Bochs, QEMU, Paravellis, Microsoft Virtual Server, Java JVM, C#/.NET CLR</td>
<td>Xen, UML, Denali</td>
<td>Solaris 10 “Zones” (Oracle Solaris 10), OpenVZ, Virtuozzo, Linux Virtual Server, FreeBSD-Jails</td>
</tr>
<tr>
<td>Transmeta Crusoe processor (still on VLIW – more power efficient)</td>
<td>Transmeta Crusoe processor (still on VLIW – more power efficient)</td>
<td>Transmeta Crusoe processor (still on VLIW – more power efficient)</td>
</tr>
</tbody>
</table>

Paravirtualization Example: Xen

- Control Plane Software
  - User software
    - Guest OS (XenLinux)
    - Guest OS (XenBSD)
    - Guest OS (XenXP)
  - User software
    - Guest OS (XenLinux)
    - Guest OS (XenBSD)
    - Guest OS (XenXP)
  - User software
    - Guest OS (XenLinux)
    - Guest OS (XenBSD)
    - Guest OS (XenXP)

Hardware (SMP x86, MMU, physical memory, network access, SCSI/HDD)

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

- Guest programs
  - CPU resources
    - Memory resources
    - Network addresses
  - Server programs
    - CPU resources
    - Memory resources
    - Network addresses
- Zone Management
  - Solaris Kernel
  - Network addresses
- Devices

The Java Virtual Machine

- Java program class files
  - JVM
    - Java interpreter
    - host system (Windows, Linux, etc.)
- Java API class files

All accesses to system resources mediated by JVM → “Sandbox”
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.
- And many others...