

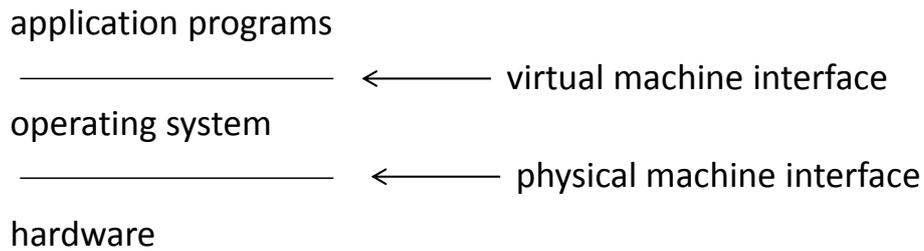
Case studies

- EECS 482: Introduction to operating systems **principles**
 - Abstraction
 - Management of shared resources
 - Indirection
 - Concurrency
 - Atomicity
 - Protection
 - Naming
 - Security
 - Reliability
 - Scheduling
 - Fairness
 - Performance

Two case studies

- Virtual machines (e.g., VMware)
 - Software layer below the operating system
- Web browsers (e.g., Google Chrome)
 - Software layer above the operating system

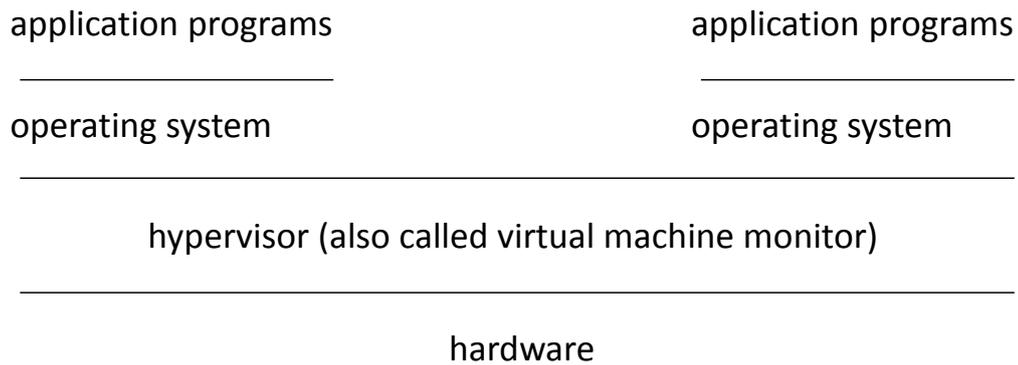
Virtual machines (e.g., VMware)



- Operating systems manage the computer and provide abstractions for application programs
- But who manages the computer and provides abstractions for the operating system?

What type of management or abstractions does an operating system need?

System structure with virtual machines



- How does this work?

Operating system virtualizes the computer for applications

- OS could intercept and simulate each instruction
 - E.g., simulate each load/store, translate virtual address to physical address (similar to `vm_syslog`)
 - Slow
- Instead, OS allows most instructions to run directly on the hardware
 - Requires hardware support (MMU)
- Not all instructions can run directly on the hardware
 - Some instructions are privileged (e.g., I/O instructions)
 - Privileged instructions should trap to the OS. OS can then kill the (usually misbehaving) application.

Hypervisor virtualizes the computer for operating systems

- Hypervisor could do this by intercepting and simulating each instruction
 - E.g., hypervisor wants to partition 2 GB physical memory into two 1 GB partitions
 - OS #2 issues what it thinks is a physical memory address. Hypervisor intercepts and translates by adding 1 GB to address
 - Slow
- Instead, hypervisor lets most instructions pass through and run directly on the hardware
- How should hypervisor handle privileged instructions issued by OS?

OS virtualization vs. hypervisor virtualization

- Compatibility
 - Hypervisor tries to provide backward compatibility for operating systems that normally run directly on hardware
 - OS doesn't need to support applications that run directly on hardware
- Abstractions
 - Hypervisor is trying to provide the illusion (to operating systems) of running directly on the hardware
 - OS is trying to provide the illusion of running on a different (nicer) computer
 - E.g., threads are nicer than processors
 - E.g., file systems are nicer than disks

Complication #1: x86 is not classically virtualizable

- Run OS in user mode, but make it look (to the OS) like it's running in kernel mode, directly on the hardware
- All instructions that behave differently in user mode and kernel mode should fault to the hypervisor, so the hypervisor can simulate running directly on the hardware
- Unfortunately, some x86 instructions behave differently in user and kernel mode, but do NOT fault when run in kernel mode
- How to handle this?

Complication #2: need multiple levels of memory translation

- Operating system uses MMU to translate from virtual address to “physical” memory
- Hypervisor would like to use MMU to translate from the OS’s “physical memory” to the real physical memory (sometimes called “machine memory”)
- Would like two levels of translation: virtual -> physical -> machine
- How to provide multiple levels of translation?

Do we need both hypervisors and operating systems?

- Hypervisor does partitioning; OS handles abstractions?

Google Chrome

- <http://www.google.com/googlebooks/chrome>
- What's a web browser?
 - An important application
 - An environment for displaying and running web pages
- A browser hosts web pages, just like an operating system hosts applications
- Important properties for browsers
 - Stability
 - Speed
 - Security

Concurrency

- Early browsers were single threaded
 - Asynchronous → inconvenient
 - Blocking → unsafe
- Using multiple threads allows the browser to be convenient (use blocking interface) and safe (single thread can't hang the system)
- Multi-threaded versus multi-process
 - What's the difference?

- What are the implications of each window having its own address space?

Chrome processes

- Chrome process manager
 - Process manager == operating system
- Process isolation
 - Separate processes → separate threads
 - One thread can block without blocking other threads
 - Can kill one tab without killing other tabs
 - What kind of bugs are they worried about?

- Do they survive all bugs?

Memory management

- Creating a new process costs more than creating a new thread
 - But Google claims that using processes leads to less memory bloat
- One address space for entire browser
 - Ideally, closing a tab frees all memory for that tab
 - But not all memory is cleaned up when a tab closes → fragmentation
 - Why is this a problem?
- One address space per browser window
 - Destroying entire process guarantees that you free all memory for that process
 - Takes advantage of the fact that windows are relatively independent
- Memory manager == operating system

Security

- Assume browser will be compromised. How to limit damage?
- Sandboxing
 - Principle of least privilege
 - Browser process can do only a few things, e.g., display to screen, respond to user communication
- Google writes browser to work with low privilege. But not all browser code is written by Google.
 - Similarly, operating systems must support third-party software (extensions), e.g., device drivers
 - Plugins run at same privilege level as main Chrome process
 - Main Chrome process is like the operating system
 - Renderer processes are like application processes
- How to protect against bad plugins
 - Rewrite to work with low privilege
 - Separate plugin process from renderer process. Similar to microkernel with multiple server processes

Phishing

- Google Chrome monitors web pages for URLs that are known to be bad
 - Similar to virus scanning

Google Gears

- Browser plugin that provides an API (application programming interface) for web pages
- Similarly, operating systems provide an API (system calls) for application programs

Testing

- Chrome Bot (Google's autograder)
- Test early; test often
- Test case design
 - Prioritize testing of popular web pages
 - Unit tests (micro test cases)
 - Macro test cases, e.g., random input (fuzz testing)

- Other topics
 - Dynamic code generation for Javascript. Not an issue for operating systems, since they usually don't virtualize the instruction set (the compiler does that)