

## Case studies

EECS 482: Introduction to operating systems?

Operating system has two main functions

- Abstraction
- Management

Operating systems are concerned with

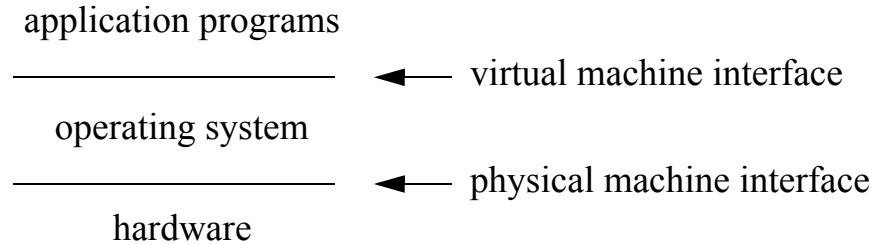
- Performance
- Fairness
- Reliability and security
- Convenience

These functions and concerns apply to many software and hardware systems beyond operating systems

Two case studies

- Virtual machines (e.g., VMware): below traditional operating systems
- Web browsers (e.g., Google Chrome): above traditional operating systems

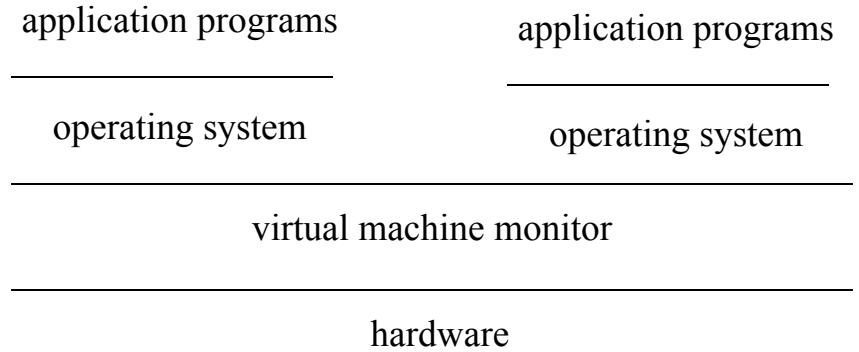
## Virtual machines (e.g., VMware)



Operating system manages computer and provides abstractions for application programs, but who manages the computer and provides abstractions for operating systems?

**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 do this by intercepting and simulating each instruction

- E.g., simulate each load/store, translate virtual address to physical address
- 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 OS. OS can then kill the (misbehaving) application.

## Virtual machine monitor virtualizes the computer for operating systems

VMM could do this by intercepting and simulating each instruction

- E.g., VMM wants to partition 2 GB physical memory into two halves
- OS #2 issues what it thinks is a physical memory address. VMM intercepts and translates by adding 1 GB to the address.
- Slow

Instead, VMM lets most instructions pass through and run directly on the hardware

How to handle privileged instructions?

How can VMM cause privileged instructions (issued by OS) to fault?

What should the VMM do when the OS issues a privileged instruction?

## **OS virtualization vs. VMM virtualization**

### Compatibility

- VMMs try to provide backward compatibility for operating systems that are used to running directly on hardware
- Operating systems are not trying to support applications that are used to running directly on hardware

### Abstractions

- VMMs are trying to provide the illusion of running on a real computer
- Operating systems are trying to provide the illusion of running on a different (nicer) computer
  - E.g., file systems are nicer than disks
  - E.g., sockets are nicer than network interface cards

## **Complication #1: x86 is not classically virtualizable**

Run OS in user mode, but make it look (to the OS) like we're running the OS in kernel mode, directly on the hardware

All instructions that return a different result in user mode and kernel mode should fault to the VMM, so the VMM can make it look like it was running in kernel mode

- Unfortunately, some x86 instructions return a different result in user mode than in kernel mode, but do NOT fault when run in user mode.

How to handle this?

## **Complication #2: need multiple levels of memory translation**

Operating system uses MMU to translate from application address space to “physical” memory

VMM wants to use the MMU to translate from the OS's “physical” memory to the real physical memory

- Sometimes the real physical memory is called “machine memory”.

Would like TWO levels of translation

## How to provide multiple levels of translation?

## Do we need both VMM and operating systems?

VMM does partitioning

Maybe run each application in its own virtual machine

- These are called “virtual appliances”



## 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

Google claims this 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 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
- E.g., 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 microkernels with multiple server processes.

## Phishing

Google Chrome monitors web pages for known-bad URLs

- 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

Types of test cases

- 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 (compiler does that).