Frequently asked questions from the previous class survey

- Belady's anomaly and local frame replacement policies?
- Multiprogramming?
Topics covered in this lecture

- Virtualization

What we will look at

- Memory and I/O Virtualization
- Type 1 and Type 2 hypervisors
- Techniques for efficiency
- Advantages
- Brief History
- Requirements
- Why?
WHY VIRTUALIZATION

Firms often have multiple, dedicated servers: e-mail, FTP, e-commerce, web, etc.

- **Load**: Maybe one machine cannot handle all that load
- **Reliability**: Management does not trust the OS to run 24 x 7 without failures
- By putting one server on a separate computer, if one of the server crashes?
  - At least the other ones are not affected
- If someone breaks into the web server, at least sensitive e-mails are still protected
  - **Sandboxing**
But ...

- While this approach achieves **isolation** and fault tolerance
  - This solution is expensive and hard to manage because so many machines are also involved
- Other reasons for having separate machines?
  - Organizations depend on more than one OS for their daily operations
    - Web server on Linux, mail server on Windows, e-commerce server on OS X, other services on various flavors of UNIX

What to do?

- A possible (and popular) solution is to use virtual machine technology
- This sounds very hip and modern
  - But the idea is old ... dating back to the 1960s
  - Even so, the way we use it today is definitely new
Main idea

- **VMM** (Virtual Machine Monitor) creates the *illusion* of multiple (virtual) machines on the same physical hardware
  - VMM is also known as a *hypervisor*
    - We will look at type 1 hypervisors (bare metal) and type 2 hypervisors (use services and abstractions offered by an underlying OS)
- **Virtualization** allows a single computer to host multiple virtual machines
  - Each potentially running a different OS

Failure in one of the virtual machines does not bring down any others

- Different servers run on different virtual machines
  - Maintains *partial-failure* model at a lower cost with easier maintainability
- Also, we can run different OS on the same hardware
  - Benefit from virtual machine isolation in the face of attacks
  - Plus enjoy other good stuff: savings, real estate, etc.
But isn’t consolidating servers like this putting all your eggs in the same basket?

- If the server running the virtual machines fails?
  - The result is even more catastrophic than the crashing of a single dedicated server

Why virtualization works [1/2]

- Service outages are due not to faulty hardware, but due to poor software, emphatically including OSes
  - Ill-designed, unreliable, buggy, and poorly configured software
Why virtualization works

- The only software running in the *highest privilege* is the hypervisor.
- Hypervisor has 2 orders of magnitude fewer lines of code than a full operating system.
  - Has 2 orders of magnitude fewer bugs.
- A hypervisor is simpler than an OS because it *does only one thing*.
  - Emulate copies of the bare metal (most commonly the Intel x86 architecture).

Advantages to running software in VMs besides strong isolation

- Few physical machines
  - Saves money on hardware and electricity
  - Takes up less rack space
- For companies such as Amazon or Microsoft
  - Reducing physical demands on data centers represents huge cost savings.
  - Companies frequently locate their data centers in the middle of nowhere.
    - Just to be close to hydroelectric dams (and cheap energy).
Price-per-kilowatt hours by region:
Easier to ship photons than electrons

<table>
<thead>
<tr>
<th>Price per KWH</th>
<th>Where</th>
<th>Possible Reasons Why</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.6¢</td>
<td>Idaho</td>
<td>Hydroelectric power; not sent long distance</td>
</tr>
<tr>
<td>10.0¢</td>
<td>California</td>
<td>Electricity transmitted long distance over the grid; Limited transmission lines in Bay Area; No coal fired electricity allowed in California.</td>
</tr>
<tr>
<td>18.0¢</td>
<td>Hawaii</td>
<td>Must ship fuel to generate electricity</td>
</tr>
</tbody>
</table>


Checkpointing and migration

- For load balancing across multiple servers
- Easier with VMs than migrating processes running on a normal OS
- Why?
  - In the bare metal case, a fair amount of critical state information about each process is kept in OS tables
  - When migrating a VM, all that has to be moved are the memory and disk images
    - All the OS tables move as well
Other uses of virtual machines

- Run legacy applications
- Software development: Test software on myriad OSes
  - No need to get a dozen computers and install a dozen OS
    - Just install a dozen VMs
    - Of course you could have partitioned hard-disk and installed a different OS but that is more difficult
      - Standard PCs allow only four primary disk-partitions, no matter how big the disk is
      - Although a multiboot program can be installed in the boot-block, it would be necessary to reboot computer to work on a new OS
  - With VMs, all of them run at once, since they are just glorified processes

Key idea of the cloud is straightforward

- Outsource computation/storage needs to a well managed data center
- Pay for use of resources, but at least you will not have to worry about physical machines, power, cooling, and maintenance
A BRIEF HISTORY OF VIRTUALIZATION

November 13, 2018

1960s

- Early 1960s IBM experimented with not just one, but two independently developed hypervisors
  - SIMMON and CP-40

- CP-40 was a research project that was reimplemented as CP-67 to form the control program of CP/CMS a virtual machine OS for IBM/360
1970s

- In 1974, Gerald Popek and Robert Goldberg published a seminal paper*
  - Listed what conditions a computer architecture should satisfy to support virtualization efficiently
- Famously, the well-known x86 architecture that originated in the 1970s did not meet this for decades
- 1970s were very productive, seeing the birth of UNIX, Ethernet, Cray-1, Microsoft, and Apple


The path to VMware

- Researchers at Stanford developed a new hypervisor called Disco
  - Went on to found VMware a virtualization giant
    - Offers type 1 and type 2 hypervisors
- VMware introduced its first virtualization solution for x86 in 1999
- Other products followed in its wake
  - Xen, KVM, VirtualBox, Hyper-V, Parallels
Requirements for Virtualization

Trap: Revisiting the concept

- A **trap** is a synchronous interrupt caused by an exceptional condition
  - E.g.: divide by zero, invalid memory access, etc.

- Usually results in a **switch to kernel mode**
  - The kernel performs some action before returning control to the originating process
Requirements for virtualization

- Virtual machines must act just like the real McCoy
  - Must be possible to boot them and install arbitrary OS on them
    - Just as on the real hardware

- Task of the hypervisor is to provide this illusion and to do it efficiently

Hypervisors should score well on

- Safety
  - Hypervisor should have full control of the virtualized resources

- Fidelity
  - Behavior of program on a virtual machine should be identical to the same program running on bare hardware

- Efficiency
  - Much of the code in the virtual machine should run without intervention from the hypervisor
Safety

- Consider each instruction in turn in an interpreter (such as Bochs) and perform exactly what is needed
  - May execute some instructions (INC) as is, but other instructions must be simulated
- We cannot allow the guest OS to disable interrupts for the entire machine or modify page-table mappings
  - Trick is to make the guest OS believe that it has
- Interpreter may be safe, even hi-fi, but performance is abysmal
  - So, VMMs try to execute most code directly

Fidelity

- **Privileged** instructions
  - Trap if the processor is in user mode and do not trap if it is in system mode (supervisor mode)
- **Control sensitive** instructions
  - Attempt to change configuration of system resources
- **Behavior sensitive** instructions
  - Whose behavior or result depends on the configuration of resources (content of relocation register or processor's mode)

A machine is virtualizable only if sensitive instructions are a subset of privileged instructions
Virtualization has long been a problem on x86
- Defects in 386 carried forward into new CPUs for 20 years in the name of backward compatibility

If you do something in user mode that you should not
- The hardware should trap!
- IBM/370 had this property, Intel’s 386 did not

Several sensitive 386 instructions were ignored if executed in user mode
- Or executed with a different behavior
- E.g. POPF instruction replaces flags register which changes the bit that enables/disables interrupts
  - In user-mode this bit was simply not changed

Also, some instructions could read sensitive state in user mode without causing a trap
Fidelity and the x86

- The x86 contained 18 sensitive, unprivileged instructions

- Sensitive register instructions
  - Read or change sensitive registers or memory locations such as a clock register or interrupt registers

- Protection system instructions
  - Reference the storage protection system, memory or address relocation system

Problem solved in 2005

- When Intel and AMD introduced virtualization in their CPUs
  - Intel CPUs: It is called VT (Virtualization Technology)
  - AMD CPUs: SVM (Secure Virtual Machine)

- Create containers in which VMs can be run

- When a guest OS is started in a container, continues to run until it causes an exception and traps to the hypervisor
  - For e.g. by executing an I/O instruction

- Set of operations that trap is controlled by a hardware bit map set by hypervisor
  - Classical trap-and-emulate approach becomes possible
What happened before that?

- Hypervisors before 2005 did not really run the original guest OS
  - Rewrote part of the code on the fly
    - To replace problematic instructions with safe code sequences that emulated original instruction
    - Replace instructions that are sensitive but not privileged
    - Binary Translation

Full virtualization

- Trap all instructions
- Fully simulate entire computer
- Trade-off: High overhead
- Benefit: Can virtualize any OS
Paravirtualization [1/2]

- Never aims to present a virtual machine that looks just like the actual underlying hardware

- Present **machine-line software interface** that explicitly exposes that it is a virtualized environment
  - Offers a set of **hypercalls** that allow the guest to send explicit requests to the hypervisor
  - Similar to how a system call offers kernel services to applications

- **DRAWBACK:** Guest OS has to be aware of the virtual machine API

Paravirtualization [2/2]

- Guests use hypercalls for privileged, sensitive operations like updating page tables
  - But they do it in cooperation with the hypervisor
  - Overall system can be simpler and faster

- Paravirtualization was offered by IBM since 1972

- Idea was revived by Denali (2002) and Xen (2003) hypervisors
Not all virtualization attempt to trick the guest into believing it has entire system

- Sometimes the aim is allow a process to run that was run on different OS and/or architecture
  - **Process-level virtualization**

- **Examples:**
  - WINE Compatibility layer allows Windows applications to run on POSIX-compliant systems like Linux, BSD, OS X
  - Process-level version of the QEMU emulator allows applications for one architecture to run on another
Terms

- **Guest Operating System**
  - The OS running on top of the hypervisor

- **Host Operating System**
  - For a type 2 hypervisor: the OS that runs on the hardware

- **Safe executions**
  - Execute the machine's instruction set in a safe manner
  - Guest OSes may change or mess up its own page tables … but not those of others

Type 1 hypervisor

- Only program running in the most privileged mode
- Support multiple copies of the actual hardware
  - Virtual machines
  - Similar to processes a normal OS would run
Location of Type-1 hypervisor

- Excel
- Word
- Emacs

- Windows
- Linux

- Control Domain

- Type 1 hypervisor

- Hardware
  (CPU, disk, network, interrupts, etc)

Control Domain  in the Type-1 hypervisor: Also known as Dom0

- Is a VM like the guest VMs, with two functional differences
  - Has the ability to talk to the hypervisor to instruct it to start and stop guest VMs
  - By default contains the device drivers needed to address the hardware
Type 2 hypervisor

- Also referred to a **hosted hypervisor**
- Relies on a host OS, say Windows or Linux, to allocate and schedule resources
- Still pretends to be a full computer with a CPU and other devices

---

Type 2: Running Guest OS

- When it starts for the first time, acts like a newly booted computer
  - Expects to find a DVD, USB drive or CD-ROM containing an OS
    - The drive could be a virtual device
    - Store the image as an ISO file on the hard drive and have hypervisor pretend its reading from proper DVD drive
  - Hypervisor installs the OS to its virtual disk (just a file) by running installation that it found on DVD
  - Once guest OS is installed on virtual disk, it can be booted and run
Location of Type-2 hypervisor

Examples of hypervisors [Partial List]

<table>
<thead>
<tr>
<th>Virtualization Method</th>
<th>Type 1 hypervisor</th>
<th>Type 2 hypervisor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtualization without hardware support</td>
<td>ESX Server 1.0</td>
<td>VMware workstation 1.0</td>
</tr>
<tr>
<td>Paravirtualization</td>
<td>Xen 1.0</td>
<td></td>
</tr>
<tr>
<td>Virtualization with hardware support</td>
<td>vSphere, Xen, Hyper-V</td>
<td>VMware Fusion, KVM, Parallels</td>
</tr>
<tr>
<td>Process Virtualization</td>
<td></td>
<td>WINE</td>
</tr>
</tbody>
</table>
The contents of this slide-set are based on the following references

- https://en.wikipedia.org/wiki/Popek_and_Goldberg_virtualization_requirements