Frequently asked questions from the previous class survey

- Type-2 hypervisor and the OS interactions?
- ARM: Advanced RISC Machine
- Is the VM a software thing?

Topics covered in this lecture

- Techniques for efficient virtualization
- Virtualizing the unvirtualizable
- Cost of virtualization
- Memory virtualization
- Virtual Appliances

Type-1 hypervisors

- Virtual machine runs as a user-process in user mode
- Not allowed to execute sensitive instructions (in the Popek-Goldberg sense)
- But the virtual machine runs a Guest OS that thinks it is in kernel mode (although, of course, it is not)
- Virtual kernel mode
- The virtual machine also runs user processes, which think they are in the user mode
- And really are in user mode

Modes
Execution of kernel model instructions
- What if the Guest OS executes an instruction that is allowed only when the CPU is really in kernel mode?
  - On CPUs without VT (Intel: Virtualization Technology)?
    - Instruction fails and the OS crashes
  - On CPUs with VT?
    - A trap to the hypervisor does occur
      - Hypervisor can inspect instruction to see if it was issued:
        - By Guest OS: Arrange for the instruction to be carried out
        - By user-process in that VM: Emulate what hardware would do when confronted with sensitive instruction executed in user-mode

We delight in the beauty of the butterfly, but rarely admit the changes it has gone through to achieve that beauty. — Maya Angelou

VIRTUALIZING THE UNVIRTUALIZABLE

Virtualizing the x86 before VT (and AMD SVM)
- Virtualizing is straightforward when VT is available
- When it is not available?
  - Make clever use of:
    1. Binary translation
    2. Hardware features that did exist on the x86

Protection rings
- The x86 supported 4 protection modes (or rings)
  - Ring 4 is the least privileged
  - This is where normal processes execute
  - You cannot execute privileged instructions
  - Ring 0 is the most privileged
  - Allows execution of any instruction
  - In normal operation, the kernel runs here
  - Other rings were never used by operating systems

x86 privilege level architecture without virtualization

In other words, hypervisors had some room to play with
- Many solutions kept the hypervisor in kernel mode (ring 0)
- Applications in user mode (ring 3)
- Guest OS in a layer of intermediate privilege (ring 1)
How this allows virtualization ...

- Kernel is privileged relative to user processes
- Any attempt to access kernel memory from a user program leads to an access violation
- Guest OS' privileged instructions trap to the hypervisor
- Hypervisor performs sanity checks and then performs instructions on the guest's behalf

But what about sensitive instructions in the guest OS' kernel code?

- The hypervisor makes sure that they no longer exist
  - Hypervisor rewrites code one basic block at a time
- Basic block
  - Short, straight-line sequences that end with a branch
  - Contain no jump, call, trap, return or other instructions that alter flow of control
  - Except for the very last instruction which does precisely that

Dynamic translation and emulation sound very expensive

- But typically are not cached
- Translated blocks are cached
  - So no translation is needed in the future
- After basic block has completed executing, control is returned to hypervisor
  - Which locates block's successor
  - If successor has already been translated, it can be executed immediately

Using the x86 rings prior to VT/SVM

- Prior to executing a basic block, hypervisor scans it to see if there are sensitive instructions
  - If so, replace with call to hypervisor procedure that handles them

Executing basic blocks

- Common to perform binary translation on all the guest OS code running in ring 1
- Replace even the privileged, sensitive instructions that could be made to trap
  - Traps can be expensive and binary translation leads to better performance

Binary translations
What about Type 2 hypervisors?

- Though type 2 hypervisors are conceptually different from type 1
  - They use, by and large, the same techniques
  - For example, VMware ESX Server (type 1, 2001) used exactly the same binary translation as the first VMware Workstation (type 2, 1999)

For faithful virtualization

- Guest OS should also be tricked into thinking it is the true and only king/queen of the mountain
  - Full control of all machine’s resources
  - Access to entire address space (4GB on 32-bit machines)
- When the queen finds another king squatting in its address space?

Let’s look at this 2 kings/queen problem

- In Linux, a user process has access to just 3 GB of the 4 GB address space (32-bit addressing)
  - 1 GB is reserved for the kernel
  - Any access to kernel memory leads to a trap
- We could take the trap and emulate appropriate actions
  - Expensive

Type 2 hypervisors have a kernel module operating in ring 0

- Allows manipulation of hardware with privileged instructions
  - Allows the guest to have the full address space
- This is all well and good, but …
  - At some point hypervisor needs to clean up and restore original processor context

What if the guest is running and an interrupt arrives from an external device?

- Type 2 hypervisor depends on host’s device drivers to handle to the interrupt
- So, the hypervisor reconfigures hardware to to run the host OS system code
  - When the device driver runs, it finds everything just as it expected it to be
- Hypervisor behaves just like teenagers throwing a party when parents are away
  - It’s OK to rearrange furniture completely, as long as they put it back as they found it before parents get home

World switch

- Going from a hardware configuration for the host kernel to a configuration for the guest OS
Why do hypervisors work even on unvirtualizable hardware?

- Sensitive instructions in the guest kernel replaced by calls to procedures that emulate these instructions.
- No sensitive instructions issued by the guest OS are ever executed directly by true hardware. Instead, they are turned into calls to the hypervisor, which emulates them.

Cost of virtualization

- We expect CPUs with VT would greatly outperform software techniques.
- Trap-and-emulate approach used by VT hardware generates a lot of traps and these are expensive. They ruin CPU caches, TLBs, and branch predictions.
- In contrast, when sensitive instructions are replaced by calls to hypervisor procedures, none of this context-switching overhead is incurred.

Still ... with modern VT hardware, usually the hardware beats the software.

True virtualization & paravirtualization

- True virtualization: Unmodified Windows, Trap due to sensitive instruction.
- Paravirtualization: Modified Linux, Trap due to hypervisor call.

To Summarize

- True virtualization: Type 1 hypervisor, Hardware.
- Paravirtualization: Microkernel, Trap due to sensitive instruction, Trap due to hypervisor call.
x86 privilege level architecture without virtualization

- Ring 0: Direct execution of User and OS Requests
- Ring 1: Direct execution of User and OS Requests
- Ring 2: Direct execution of User and OS Requests
- Ring 3: Direct execution of User and OS Requests

Full Virtualization: Binary translation approach to x86 virtualization

- VMM acts as a translator
- Guest OS is modified
- OS requests are translated

Paravirtualization approach to x86 virtualization

- Guest OS is unmodified
- Guest OS issues "hypercalls" to the virtualization layer

Hardware assisted virtualization

- OS requests trap to VMM
- No binary translation or paravirtualization

Contrasting the virtualization approaches

<table>
<thead>
<tr>
<th>Technique</th>
<th>Full virtualization with Binary Translation</th>
<th>Hardware Assisted Virtualization</th>
<th>OS Assisted Virtualization/Paravirtualization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guest Modification/Compatibility</td>
<td>Binary Translation and Direct Execution</td>
<td>Exit to Boot Mode on privileged instructions</td>
<td>Hypercalls</td>
</tr>
<tr>
<td></td>
<td>Unmodified Guest OS</td>
<td>Unmodified Guest OS</td>
<td>GuestOS modified to issue hypercalls so it can’t run on native hardware.</td>
</tr>
<tr>
<td></td>
<td>Excellent compatibility</td>
<td>Excellent compatibility</td>
<td>Compatibility is lacking.</td>
</tr>
</tbody>
</table>

Memory Virtualization

- VMM abstracts memory access
All modern OS support virtual memory

- Basically **mapping** of virtual address space onto pages of physical memory
- Defined by (multilevel) page tables
- Mapping is set in motion by having the OS set a control register that points to the top-level page table
- Virtualization greatly complicates memory management

Scenario

- Guest OS decides to map its virtual pages 7, 4, and 3 onto physical pages 10, 11, and 12 respectively
- Builds page tables and sets hardware register to point to top level page table
- Sensitive instruction that traps on a VT CPU
- We will look at type 1 but the problem is the same in type 2 and paravirtualization

What should the hypervisor do?

- Allocate physical pages 10, 11, and 12 to the VM
  - Setup page tables to map VM’s virtual pages 7, 4, 3
- What if a second VM starts up and maps its virtual pages 4, 5, and 6 to physical pages 10, 11 and 12?
  - This VM loads a control register to point to its page tables
  - Hypervisor catches this trap

Choices for the hypervisor

- Cannot use the mapping from the 2nd VM because physical pages 10, 11, and 12 are already in use
- Find free pages, say 20, 21, and 22 and use them
  - But first, create new page tables mapping virtual pages 4, 5, and 6 of VM-2 onto 20, 21, and 22
- In general for each VM, the hypervisor needs to create a **shadow page table**
  - Map virtual pages used by VM onto actual physical pages that the hypervisor gave it

Also...

- Every time the Guest OS changes its page tables?
  - The hypervisor must change the shadow page tables as well
  - If the guest OS remaps virtual page 7 onto what it sees as physical page 200
    - The hypervisor has to know about this change
  - Trouble is that the guest OS can change its page tables by just writing into memory
    - No sensitive operations are required, so the hypervisor does not even know about the change
    - Certainly cannot update shadow page tables used by actual hardware

Options

- Keep track of the top-level page table
  - There is a trap when the guest OS attempts to load register
    - Map the page tables it points to as read-only
      - If the guest OS tries to modify it, will cause a fault and give control to the hypervisor
      - Figure out what the guest OS is trying to do and update shadow tables accordingly
  - Allow guest to add new mappings at will
    - Nothing changes in the shadow tables
    - When a new page is accessed, fault occurs and control reverts to hypervisor (can then add entries)
Hardware support for nested page tables

- Took AMD and Intel a few years to produce hardware to virtualize memory efficiently
- Support for nested page tables (AMD)
  - Intel calls this extended page tables (EPT)
  - With EPT
    - Hypervisor still has the shadow page table, but CPU is able to handle intermediate levels in hardware
    - Hardware walks the EPT to translate guest virtual address to guest physical address
    - Also, walks the EPT to find the host physical address without software intervention

Other issues

- Overcommitment of physical memory
  - 1 physical machine with 32 GB of memory will run 3 VMs each of which thinks there is 16 GB of memory

- Deduplication
  - Allow sharing of pages with the same content
  - E.g. Linux kernel

How can we take away memory pages safely from VMs?

- There is a trick known as ballooning
- Small balloon module loaded into each VM as a pseudo device driver that talks to hypervisor
- Balloon inflates at hypervisor’s request by allocating more and more pinned pages
- And deflates by deallocating these pages

How ballooning helps

- As balloon inflates
  - Memory scarcity in the guest increases
  - The guest OS responds by paging out what it believes are the least valuable pages
    - This is exactly what we need!
  - As balloon deflates
    - More memory available for the guest to allocate

In other words

- Hypervisor tricks the guest OS into making tough decisions for it
- In politics this is known as passing the buck

Virtual Appliances

...
Installing application software

- VMs offer a solution to a problem that has long plagued users (especially open source)
  - How to install application programs
- Applications are dependent on numerous other applications and libraries
  - Which themselves depend on a host of software packages
- Plus there are dependencies on particular versions of compilers, scripting languages, OS etc.

With VMs …

- Developer can carefully construct a virtual machine
  - Load it with required OS, compiler, libraries, and application code
  - Freeze the entire unit … ready to run
- Only the software developer has to understand the dependencies

What about customers?

- Customers get a complete package that actually works
  - Completely independent of which OS they are running and which other software, packages, and libraries they have
- These are “shrink-wrapped” virtual machines
  - Virtual appliances
- Amazon’s EC2 cloud offers many pre-packaged virtual appliances
  - Software as a service

Clouds

- Virtualization played a critical role in the dizzying rise of cloud computing
- Clouds
  - Public or private or federated
- Clouds offer different things
  - Bare metal
  - VMs of different sizes and capabilities
  - Appliances with software that is ready to use

5 characteristics of clouds: NIST

- On-demand self-service
  - No human interaction needed
- Broad network access
  - Resources available over the network
  - Resource pooling
  - Resources pooled among multiple users
- Rapid elasticity
  - Acquire and release resources rapidly
- Measured service
  - Meters resource usage
Licensing Issues

- Some software is licensed on a per-CPU basis
  - Especially, software for companies
  - When they buy a program they have the right to run it on just one CPU
  - What is a CPU anyway?
  - Can we run multiple VMs all running on the same physical hardware?
- Problem is even worse, when companies have licenses for N machines running the software
  - VMs come and go on demand

The contents of this slide-set are based on the following references

- VMWare: Understanding Full Virtualization, Paravirtualization, and Hardware Assist.