CSCE 410/611: Virtualization

- Definitions, Terminology
- Why Virtual Machines?
- Mechanics of Virtualization
- Virtualization of Resources (Memory)

- Some slides made available Courtesy of Gernot Heiser, UNSW.

Virtual Machines

- "A virtual machine (VM) is an efficient, isolated duplicate of a real machine"
- Duplicate: VM should behave identically to the real machine
  - Programs cannot distinguish between execution on real or virtual hardware
  - Except for:
    - Fewer resources available (and potentially different between executions)
    - Some timing differences (when dealing with devices)
- Isolated: Several VMs execute without interfering with each other
- Efficient: VM should execute at a speed close to that of real hardware
  - Requires that most instruction are executed directly by real hardware
Virtual Machines, Simulators and Emulators

**Simulator**
- Provides a *functionally accurate* software model of a machine
- May run on any hardware
- Is typically slow (order of 1000 slowdown)

**Emulator**
- Provides a *behavioural* model of hardware (and possibly S/W)
- Not fully accurate
- Reasonably fast (order of 10 slowdown)

**Virtual machine**
- Models a machine exactly and efficiently
- Minimal slowdown
- Needs to be run on the physical machine it virtualizes (more or less)

---

Types of Virtual Machines

> Contemporary use of the term VM is more general
> Call virtual machines even if there is no correspondence to an existing real machine
  - E.g: *Java virtual machine*
  - Can be viewed as virtualizing at the ABI level
  - Also called *process VM*
> We only concern ourselves with virtualizing at the ISA level
  - ISA = *instruction-set architecture* (hardware-software interface)
  - Also called *system VM*
  - Will later see subclasses of this
Virtual Machine Monitor (VMM), aka Hypervisor

- Program that runs on real hardware to implement the virtual machine
- Controls resources
  - Partitions hardware
  - Schedules guests
  - Mediates access to shared resources
    - e.g. console
  - Performs **world switch**
- Implications:
  - Hypervisor executes in **privileged** mode
  - Guest software executes in **unprivileged** mode
  - **Privileged instructions** in guest cause a trap into hypervisor
  - Hypervisor interprets/emulates them
  - Can have extra instructions for **hypercalls**

Why Virtual Machines?

- Historically used for easier sharing of expensive mainframes
  - Run several (even different) OSes on same machine
  - Each on a subset of physical resources
  - Can run single-user single-tasked OS in time-sharing system
    - legacy support
    - "world switch" between VMs
- Gone out of fashion in 80’s
  - Time-sharing OSes common-place
  - Hardware too cheap to worry...
**Why Virtual Machines?**

- Renaissance in recent years for improved isolation
- Server/desktop virtual machines
  - Improved QoS and security
  - Uniform view of hardware
  - Complete encapsulation
    - replication
    - migration
    - checkpointing
    - debugging
  - Different concurrent OSes
    - e.g.: Linux and Windows
  - Total mediation
- Would be mostly unnecessary
  - if OSes were doing their job...

---

**Uses of Virtual Machines**

- Multiple (identical) OSes on same platform
  - the original *raison d’être*
  - these days driven by server consolidation
  - interesting variants of this:
    - different OSes (Linux + Windows)
    - old version of same OS (Win2k for stuff broken under Vista)
    - OS debugging (most likely uses Type-II VMM)
- Checkpoint-restart
  - minimise lost work in case of crash
  - useful for debugging, incl. going backwards in time
  - re-run from last checkpoint to crash, collect traces, invert trace from crash
  - life system migration
  - load balancing, environment take-home
- Ship application with complete OS
  - reduce dependency on environment
  - “Java done right” 😊
- How about embedded systems?

---

©2006 Geraint Heiser UNSW/UNICT/ACRC. Distributed under Creative Commons Attribution License.
CSCE 410/611: Virtualization

- Definitions, Terminology
- Why Virtual Machines?
- Mechanics of Virtualization
- Virtualization of Resources (Memory)

- Some slides made available Courtesy of Gernot Heiser, UNSW.
Techniques in Classical Virtualization

- **De-privileging** ("trap-and-emulate")
  - All instructions that read/write privileged state trap when executed in unprivileged level.
  - Execute guest OS directly, but at unprivileged level.
- **Para-Virtualization**
  - "Modify guest operating system to provide higher-level information to VMM."
- **Interpretive Execution**
  - Add dedicated HW execution mode for running the guest OS.
  - e.g. IBM 370 SIE ("start interpretive execution") instruction.
  - Reduces number of required traps.
- **Binary Translation**
  - WMWare

Virtualization Mechanics

- Traditional "trap and emulate" approach:
  - guest attempts to access physical resource
  - hardware raises exception (trap), invoking hypervisor's exception handler
  - hypervisor emulates result, based on access to virtual resource
- Most instructions do not trap
  - makes efficient virtualization possible
  - requires that VM ISA is (almost) same as physical processor ISA

<table>
<thead>
<tr>
<th>Guest</th>
<th>VMM</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ld r9, curr_thrd</code></td>
<td><code>lda r1, vm_reg_ctx</code></td>
</tr>
<tr>
<td><code>ld r1, (r9, ASID)</code></td>
<td><code>ld r2, (r1, ofs r0)</code></td>
</tr>
<tr>
<td><code>mv CPU ASID, r1</code></td>
<td><code>sto r2, (r1, ofs ASID)</code></td>
</tr>
<tr>
<td><code>ld sp, (r1, kern_stk)</code></td>
<td></td>
</tr>
</tbody>
</table>

©2008 Gerhard Heiser UNSW/MCTA/CCL Distributed under Creative Commons Attribution License
Virtualization has a Long History …

Formal Requirements for Virtualizable Third Generation Architectures

Gerald J. Popk.
University of California, Los Angeles
Robert P. Goldberg
Horowitz Information Systems and Harvard University

Virtual machine systems have been implemented on a limited number of third generation computer systems, i.e., the IBM 360/370, the DEC PDP-11, and the HP 1000. In all these systems, it is known that virtual memory is a virtual machine. In this paper, models of a virtual machine are presented. Formal requirements for virtual machine systems are concluded to test whether such an architecture can support virtual machine systems.

Virtual machine systems have been implemented in a limited number of third generation computer systems, i.e., the IBM 360/370, the DEC PDP-11, and the HP 1000. In all these systems, it is known that virtual memory is a virtual machine. In this paper, models of a virtual machine are presented. Formal requirements for virtual machine systems are concluded to test whether such an architecture can support virtual machine systems.

Virtual machine systems have been implemented on a limited number of third generation computer systems, i.e., the IBM 360/370, the DEC PDP-11, and the HP 1000. In all these systems, it is known that virtual memory is a virtual machine. In this paper, models of a virtual machine are presented. Formal requirements for virtual machine systems are concluded to test whether such an architecture can support virtual machine systems.

Virtual machine systems have been implemented on a limited number of third generation computer systems, i.e., the IBM 360/370, the DEC PDP-11, and the HP 1000. In all these systems, it is known that virtual memory is a virtual machine. In this paper, models of a virtual machine are presented. Formal requirements for virtual machine systems are concluded to test whether such an architecture can support virtual machine systems.

Virtual machine systems have been implemented on a limited number of third generation computer systems, i.e., the IBM 360/370, the DEC PDP-11, and the HP 1000. In all these systems, it is known that virtual memory is a virtual machine. In this paper, models of a virtual machine are presented. Formal requirements for virtual machine systems are concluded to test whether such an architecture can support virtual machine systems.

Virtual machine systems have been implemented on a limited number of third generation computer systems, i.e., the IBM 360/370, the DEC PDP-11, and the HP 1000. In all these systems, it is known that virtual memory is a virtual machine. In this paper, models of a virtual machine are presented. Formal requirements for virtual machine systems are concluded to test whether such an architecture can support virtual machine systems.

Virtual machine systems have been implemented on a limited number of third generation computer systems, i.e., the IBM 360/370, the DEC PDP-11, and the HP 1000. In all these systems, it is known that virtual memory is a virtual machine. In this paper, models of a virtual machine are presented. Formal requirements for virtual machine systems are concluded to test whether such an architecture can support virtual machine systems.

Virtual machine systems have been implemented on a limited number of third generation computer systems, i.e., the IBM 360/370, the DEC PDP-11, and the HP 1000. In all these systems, it is known that virtual memory is a virtual machine. In this paper, models of a virtual machine are presented. Formal requirements for virtual machine systems are concluded to test whether such an architecture can support virtual machine systems.

Virtual machine systems have been implemented on a limited number of third generation computer systems, i.e., the IBM 360/370, the DEC PDP-11, and the HP 1000. In all these systems, it is known that virtual memory is a virtual machine. In this paper, models of a virtual machine are presented. Formal requirements for virtual machine systems are concluded to test whether such an architecture can support virtual machine systems.

Virtual machine systems have been implemented on a limited number of third generation computer systems, i.e., the IBM 360/370, the DEC PDP-11, and the HP 1000. In all these systems, it is known that virtual memory is a virtual machine. In this paper, models of a virtual machine are presented. Formal requirements for virtual machine systems are concluded to test whether such an architecture can support virtual machine systems.

Virtual machine systems have been implemented on a limited number of third generation computer systems, i.e., the IBM 360/370, the DEC PDP-11, and the HP 1000. In all these systems, it is known that virtual memory is a virtual machine. In this paper, models of a virtual machine are presented. Formal requirements for virtual machine systems are concluded to test whether such an architecture can support virtual machine systems.

Virtual machine systems have been implemented on a limited number of third generation computer systems, i.e., the IBM 360/370, the DEC PDP-11, and the HP 1000. In all these systems, it is known that virtual memory is a virtual machine. In this paper, models of a virtual machine are presented. Formal requirements for virtual machine systems are concluded to test whether such an architecture can support virtual machine systems.

Virtual machine systems have been implemented on a limited number of third generation computer systems, i.e., the IBM 360/370, the DEC PDP-11, and the HP 1000. In all these systems, it is known that virtual memory is a virtual machine. In this paper, models of a virtual machine are presented. Formal requirements for virtual machine systems are concluded to test whether such an architecture can support virtual machine systems.

Virtual machine systems have been implemented on a limited number of third generation computer systems, i.e., the IBM 360/370, the DEC PDP-11, and the HP 1000. In all these systems, it is known that virtual memory is a virtual machine. In this paper, models of a virtual machine are presented. Formal requirements for virtual machine systems are concluded to test whether such an architecture can support virtual machine systems.

Virtual machine systems have been implemented on a limited number of third generation computer systems, i.e., the IBM 360/370, the DEC PDP-11, and the HP 1000. In all these systems, it is known that virtual memory is a virtual machine. In this paper, models of a virtual machine are presented. Formal requirements for virtual machine systems are concluded to test whether such an architecture can support virtual machine systems.

Virtual machine systems have been implemented on a limited number of third generation computer systems, i.e., the IBM 360/370, the DEC PDP-11, and the HP 1000. In all these systems, it is known that virtual memory is a virtual machine. In this paper, models of a virtual machine are presented. Formal requirements for virtual machine systems are concluded to test whether such an architecture can support virtual machine systems.

Virtual machine systems have been implemented on a limited number of third generation computer systems, i.e., the IBM 360/370, the DEC PDP-11, and the HP 1000. In all these systems, it is known that virtual memory is a virtual machine. In this paper, models of a virtual machine are presented. Formal requirements for virtual machine systems are concluded to test whether such an architecture can support virtual machine systems.

Virtual machine systems have been implemented on a limited number of third generation computer systems, i.e., the IBM 360/370, the DEC PDP-11, and the HP 1000. In all these systems, it is known that virtual memory is a virtual machine. In this paper, models of a virtual machine are presented. Formal requirements for virtual machine systems are concluded to test whether such an architecture can support virtual machine systems.

Virtual machine systems have been implemented on a limited number of third generation computer systems, i.e., the IBM 360/370, the DEC PDP-11, and the HP 1000. In all these systems, it is known that virtual memory is a virtual machine. In this paper, models of a virtual machine are presented. Formal requirements for virtual machine systems are concluded to test whether such an architecture can support virtual machine systems.
Formal Virtualization Reqs. (2)

- **Def:** Machine State: \( S = \langle E, M, P, R \rangle \)
  - \( E \): executable storage
  - \( M \): processor mode
  - \( P \): program counter
  - \( R \): relocation-bounds register

- **Def:** Instruction \( i \) is **behavior sensitive** if there exists an integer \( x \) and states:
  (a) \( S_1 = \langle e | r, m_1, p, r \rangle \), and
  (b) \( S_2 = \langle e | r \times x, m_2, p, r \times x \rangle \), where ...

- **Intuitively**, an instruction is behavior sensitive if the effect of its execution depends on the value of the relocation-bounds register, i.e., upon its location in real memory, or on the mode.

- **Example:** load physical address!

---

Formal Virtualization Reqs. (3)

**Theorem:** “For any conventional third generation [1974] computer, a virtual machine monitor may be constructed if the set of **sensitive** instructions for that computer is a subset of the set of **privileged** instructions.”
Formal Virtualization Reqs. (4)

- "Hybrid" Virtualization (with interpreted instr's):
  - Def: Machine State: \( S = \langle E, M, P, R \rangle \)
    - \( E \): executable storage
    - \( M \): processor mode
    - \( P \): program counter
    - \( R \): relocation-bounds register
  - Def: Instruction \( i \) is user sensitive if there exists a state \( S = \langle E, \text{user}, P, R \rangle \) for which \( i \) is control sensitive or behavior sensitive.
  - Theorem: A hybrid virtual machine (HVMM) monitor may be constructed for any conventional third generation machine in which the set of user sensitive instructions are a subset of the set of privileged instructions.
  - Example: PDP-10 JRST 1 (return to user mode) is non-privileged, but supervisor control sensitive. Therefore, PDP-10 cannot host VMM, but can host HVMM.

Recap: Some Obstacles to Virtualization

- "Visibility of Privileged State"
  - e.g. Current Privilege Level is stored in code segment register.
  - Guest therefore can know that it runs in deprivileged mode.

- "Lack of Traps when Privileged Instructions run at User-Level"
  - Some privileged instructions generate NOOP in user mode rather than generating a trap.
  - e.g. "pop flags", which modifies ALU and system flags, must generate trap for VMM to intervene.
Techniques in Classical Virtualization

- **De-privileging** ("trap-and-emulate")
  - All instructions that read/write privileged state trap when executed in unprivileged level.
  - Execute guest OS directly, but at unprivileged level.

- **Para-Virtualization**
  - "Modify guest operating system to provide higher-level information to VMM."

- **Interpretive Execution**
  - Add dedicated HW execution mode for running the guest OS.
  - e.g. IBM 370 SIE ("start interpretive execution") instruction.
  - Reduces number of required traps.

- **Binary Translation**
  - WMWare
Virtualization Techniques: Paravirtualization

- Present software interface to virtual machines that is similar but not identical to that of the underlying hardware.

- Provide specially defined 'hooks' to allow the guest(s) to hand over handling of difficult portions of code to VMM.

- Requires the guest operating system to be explicitly ported for the para-API.
  - A conventional O/S distribution which is not paravirtualization-aware cannot be run on top of a paravirtualized VMM!
  - Xen solution for closed-source O/Ss: paravirtualization-aware device drivers (e.g. XenWindows64Pv project) to be installed in guest O/S.

Techniques in Classical Virtualization

- De-privileging ("trap-and-emulate")
  - All instructions that read/write privileged state trap when executed in unprivileged level.
  - Execute guest OS directly, but at unprivileged level.

- Para-Virtualization
  - "Modify guest operating system to provide higher-level information to VMM."

- Interpretive Execution
  - Add dedicated HW execution mode for running the guest OS.
  - e.g. IBM 370 SIE ("start interpretive execution") instruction.
  - Reduces number of required traps.

- Binary Translation
  - VMware
**VMware Software VMM: Binary Translation**

- Traditionally, software VMMS run very slow due to interpretation.

- **Binary Translation:**
  - Replace sensitive instructions in guest binary *on-the-fly* and replace by emulation code or hypercall.
  - **Binaries** as input, not source code.
  - **Dynamic** translation at run-time.
  - **Instruction-level** translation, not at higher ABI level.
  - Input is full x86 instruction set. Output is safe subset.

---

**Binary Translation: Simple Example**

```c
int isPrime(int a) {
    for (int i = 2; i < a; i++) {
        if (a % i == 0) return 0;
    }
    return 1;
}
```

<- small example, C code

```
isPrime:  mov %ecx, %edi ; %ecx = %edi (a)
          mov %esi, $2 ; i = 2
          cmp %esi, %ecx ; is i >= a?
          jge prime ; jump if yes
nexti:   mov %eax, %ecx ; set %eax = a
          cdq
          sign-extend
          idiv %esi
          test %edx, %edx ; is remainder zero?
          jz notPrime ; jump if yes
          inc %esi ; i++
          cmp %esi, %ecx ; is i >= a?
          jl nexti ; jump if no
prime:   mov %eax, $1 ; return value in %eax
          ret
notPrime: xor %eax, %eax ; %eax = 0
          ret
```

same code, compiled ->
Translation: Mechanics

1. read prefixes, opcodes, operands
2. stop at 12 instructions or terminating instruction (control flow)
3. translate simple instructions IDENT
4. others translated non-IDENT
5. generate compiled-code-fragment (CCF)

Translation Result

notPrime: xor %eax, %eax ; IDENT
pop %r11 ; RET
mov %gs:00f939eb8(%rip), %rcx ; spill %rcx
movzx %ecx, %r11b
jmp %gs:00f7d8e0(%r11)
Binary Translation: Observations

- This approach scales well:
  - e.g., Windows XP boot/halt translates
    - 229,347 64-bit translation units (TUs) of up to 12 instructions.
    - 23,909 32-bit TUs
    - 6,680 16-bit TUs
  - Translator captures execution trace of guest code.
    - This is good for instruction-cache locality
    - Rarely-executed code (e.g. error handling) is placed off the "hot" execution path.

Most instructions need no translation, except

- Instructions that are affected by translation, because code layout changes:
  - PC-relative addressing
  - Direct control flow (direct calls, branches, jumps)
  - Indirect control flow (jmp, call, ret)

- Privileged instructions:
  - Some instructions run faster in binary translation mode than native.
    - e.g. cli (clear interrupts) on Pentium 4 takes 60 cycles; replaced by "vcpu.flags.IF:=0".
  - Other operations (e.g. context switch) may need to call out to a runtime, with lots of overhead.
Binary Translation of User-Level Code?

- “BT is not required for safe execution of most user code on most guest operating systems.”

- Switch between BT and direct execution:
  - Use direct execution of guest in user-mode
  - Use BT for guest in kernel-mode

- This permits application to run at native speed.

---

CSCE 410/611: Virtualization

- Definitions, Terminology
- Why Virtual Machines?
- Mechanics of Virtualization
- Virtualization of Resources (Memory)

- Some slides made available Courtesy of Gernot Heiser, UNSW.
Memory Virtualization

- Note: Guest OS expects zero-based physical address space.

- In traditional system:
  \[ \text{virtual address} \rightarrow \text{physical address} \]

- In VMM system:
  \[ \text{virtual address} \rightarrow \text{physical address} \rightarrow \text{machine address} \]

- Each VM maintains pmap to translate physical pages to machine pages.
- Operations on TLB are intercepted by VMM, which prevents manipulation of the MMU by the guest.
- Mapping from virtual pages to machine pages is maintained in **shadow page table**.
  - This table is used by the CPU!
  - Is maintained consistent with physical \( \rightarrow \) machine mapping.

Shadow Page Table

Every time the guest modifies its page mapping, either by changing the content of a translation, creating a new translation, or removing an existing translation, the virtual MMU module will capture the modification and adjust the **shadow page tables** accordingly.
Issues in Page Replacement

- **Memory Over-Commitment**: What if memory requirements exceed available resources?
  - Move some “physical” memory to disk.

- **Issue 1**: How does this affect page replacement?
  - A page replacement algorithm now needs to pick
    - victim virtual machine (ok)
    - victim page (huh?! what is a good page to replace?!)

- **Issue 2**: Double-Paging Problem:
  - What can happen when we page out a “physical” page that is on disk?
    1. Guest picks “physical” page on disk as victim.
    2. In order to page it out by guest, it needs to be paged-in by VMM beforehand.
  - This causes **two** page faults per fault.

Avoiding paged-out “physical” pages

**Ballooning.** “ESX Server controls a balloon module running within the guest, directing it to allocate guest pages and pin them in “physical” memory. The machine pages backing this memory can then be reclaimed by ESX Server. Inflating the balloon increases memory pressure, forcing the guest OS to invoke its own memory management algorithms. The guest OS may page out to its virtual disk when memory is scarce. Deflating the balloon decreases pressure, freeing guest memory.” (Waldspurger, OSDI’02)
Potential Problems with Ballooning

• Ballooning works fine as long as it works.
• Ballooning drivers may be uninstalled, disabled explicitly, unavailable during booting.
• Upper levels on balloon sizes may be imposed by guest OSs.

• Solution: Fall back on basic paging mechanisms...
  – Problems?

How to Adjust Memory Allocation

• Memory allocation with unequal requirements across VMs?

• Fair allocation: e.g. Proportional Share algorithms.

• Reclaiming idle memory: idle memory tax.

• How to measure idle memory: sampling.
Memory Sharing across Virtual Machines

- Why memory sharing?
  - Eliminate redundant copies of pages.
  - This allows for more over-commitment of memory.

- Example: Transparent page sharing in Disco
  - Map multiple "physical" pages onto machine page, and mark it as copy-on-write.
  - Q: How do we know when a redundant copy has been created?
  - A: Need hooks into guest OS!

- Content-Based Page Sharing
  - Identify shareable pages by their content.
  - Agnostic about origin of generation of identical pages.
  - Use hashing to identify potentially shareable pages.

Content-Based Page Sharing in ESX Server

Content-Based Page Sharing. ESX Server scans for sharing opportunities, hashing the contents of candidate PPN 0x2868 in VM 2. The hash is used to index into a table containing other scanned pages, where a match is found with a hint frame associated with PPN 0x43f8 in VM 3. If a full comparison confirms the pages are identical, the PPN-to-MPN mapping for PPN 0x2868 in VM2 is changed from MPN 0x1096 to MPN 0x123b, both PPNs are marked COW, and the redundant MPN is reclaimed.