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

UNSW

- *"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

©2008 Gernot Heiser UNSW/NICTA/OKL. Distributed under Creative Commons Attribution License 3

Simulation, Emulation, Virtual Machine

- **Simulation**: Abstract model of a system is functionally simulated.
- **Emulation**: Hardware or software (or both) emulates the behavior of the guest in a host so that emulated behavior is close to behavior of real system.

"Simulators as high-level emulators."

- **Virtualization**: Virtualization involves simulating parts of a computer's hardware - enough for a guest operating system to run unmodified - but most operations still occur on the real hardware for efficiency reasons.

Types of Virtual Machines

UNSW

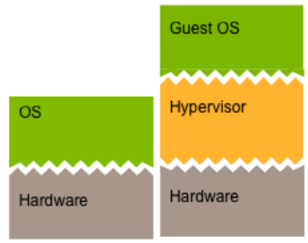
- Contemporary use of the term VM is more general
- Call virtual machines even if there is nor correspondence to an existing real machine
  - E.g: *Java virtual machine*
  - Can be viewed as virtualizing at the ABI level
  - Also called *process VM*
- We only concern ourselves with virtualizing at the ISA level
  - ISA = *instruction-set architecture* (hardware-software interface)
  - Also called *system VM*
  - Will later see subclasses of this

©2008 Gernot Heiser UNSW/NICTA/OKL. Distributed under Creative Commons Attribution License 5

Virtual Machine Monitor (VMM), aka Hypervisor

UNSW

- 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*



©2008 Gernot Heiser UNSW/NICTA/OKL. Distributed under Creative Commons Attribution License

6

CSCE 410/611: Virtualization

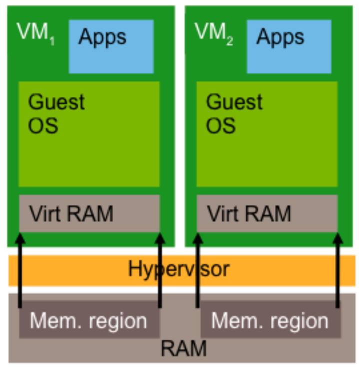
- Definitions, Terminology
- Why Virtual Machines?
- Mechanics of Virtualization
- Virtualization of Resources (Memory)

Some slides made available Courtesy of Gernot Heiser, UNSW.

Why Virtual Machines?

UNSW

- 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...

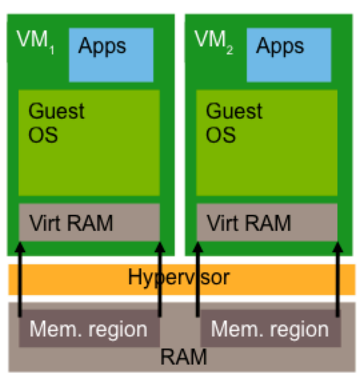


©2008 Gernot Heiser UNSW/NICTA/OKL. Distributed under Creative Commons Attribution License 7

Why Virtual Machines?

UNSW

- 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...



©2008 Gernot Heiser UNSW/NICTA/OKL. Distributed under Creative Commons Attribution License 8

Uses of Virtual Machines

UNSW

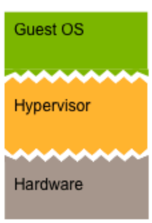
- Multiple (identical) OSes on same platform
  - the original *raison d'être*
  - these days driven by server consolidation
  - interesting variants of this:
    - different OSes (Linux + Windows)
    - old version of same OS (Win2k for stuff broken under Vista)
    - OS debugging (most likely uses Type-II 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?

©2008 Gernot Heiser UNSW/NICTA/OKL. Distributed under Creative Commons Attribution License 36

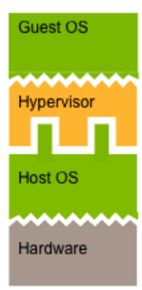
Native vs. Hosted VMM

UNSW

Native/Classic/Bare-metal/Type-I



Hosted/Type-II



- Hosted VMM can run besides native apps
  - Sandbox untrusted apps
  - Run second OS
  - Less efficient:
    - Guest privileged instruction traps into OS, forwarded to hypervisor
    - Return to guest requires a native OS system call
  - Convenient for running alternative OS environment on desktop

©2008 Gernot Heiser UNSW/NICTA/OKL. Distributed under Creative Commons Attribution License 9

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 quest 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

UNSW

- 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

Guest

```
ld   r0, curr_thrd
ld   r1, (r0,ASID)
mv   CPU_ASID, r1
ld   sp, (r1,kern_stk)
```

Exception

VMM

```
lda  r1, vm_reg_ctxt
ld   r2, (r1,ofs_r0)
sto  r2, (r1,ofs_ASID)
```

©2008 Gernot Heiser UNSW/NICTA/OKL. Distributed under Creative Commons Attribution License

11

Virtualization has a Long History …

Formal Requirements for Virtualizable Third Generation Architectures

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

Virtual machine systems have been implemented on a limited number of third generation computer systems, e.g. CP-67 on the IBM 360/67. From previous empirical studies, it is known that certain third generation computer systems, e.g. the DEC PDP-10, cannot support a virtual machine system. In this paper, model of a third-generation-like computer system is developed. Formal techniques are used to derive precise sufficient conditions to test whether such an architecture can support virtual machines.

Key Words and Phrases: operating system, third generation architecture, sensitive instruction, formal requirements, abstract model, proof, virtual machine, virtual memory, hypervisor, virtual machine monitor

CR Categories: 4.32, 4.35, 5.21, 5.22

Copyright © 1974, Association for Computing Machinery, Inc. General permission to republish, but not for profit, all or part of this material is granted provided that ACM's copyright notice is given and that reference is made to the publication, to its date of issue, and to the fact that reprinting privileges were granted by permission of the Association for Computing Machinery.

This is a revised version of a paper presented at the Fourth ACM Symposium on Operating Systems Principles, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, October 15–17, 1973.

This research was supported in part by the U.S. Atomic Energy Commission, Contract No. AT(11-1) Gen 10, Project 14 and in part by the Electronic Systems Division, U.S. Air Force, Hanscom Field, Bedford, Massachusetts under Contract Number F19628-70-0217.

Authors' addresses: Gerald J. Popek, Computer Science Department, University of California, Los Angeles CA 90024; Robert P. Goldberg, Honeywell Information Systems, Waltham, MA 02154.

Communications of the ACM — July 1974, Volume 17, Number 7

References

1. Buzen, J.P., and Gagliardi, U.O. The evolution of virtual machine architecture. Proc. NCC 1973, AFIPS Press, Montvale, N.J., pp. 291–300.
2. Gagliardi, U.O., and Goldberg, R.P. Virtualizable architectures, Proc. ACM AICA Internat. Computing Symposium, Venice, Italy, 1972.
3. Galley, S.W. PDP-10 Virtual machines. Proc. ACM SIGARCH-SIGOPS Workshop on Virtual Computer Systems, Cambridge, Mass., 1969.
4. Goldberg, R.P. Virtual machine systems. MIT Lincoln Laboratory Rept. No. MS-2686 (also 28L-0036), Lexington, Mass., 1969.
5. Goldberg, R.P. Hardware requirements for virtual machine systems. *Proc. Hawaii Internat. Conference on Systems Sciences*, Honolulu, Hawaii, 1971.
6. Goldberg, R.P. Architectural principles for virtual computer systems. Ph.D. Th., Div. of Eng. and Applied Physics, Harvard U., Cambridge, Mass., 1972.
7. Goldberg, R.P. (Ed). Proc. ACM SIGARCH-SIGOPS Workshop on Virtual Computer Systems, Cambridge, Mass., 1973.
8. Goldberg, R.P. Architecture of virtual machines. Proc. NCC 1973, AFIPS Press, Montvale, N.J., pp. 309–318.
9. IBM Corporation. IBM Virtual Machine Facility/370: Planning Guide, Pub. No. GC20-1801-0, 1972.
10. Lauer, H.C., and Snow, C.R. Is supervisor-state necessary? Proc. ACM AICA Internat. Computing Symposium, Venice, Italy, 1972.
11. Lauer, H.C., and Wyeth, D. A recursive virtual machine architecture. Proc. ACM SIGARCH-SIGOPS Workshop on Virtual Computer Systems, Cambridge, Mass., 1973.
12. Meyer, R.A., and Seawright, L.H. A virtual machine time-sharing system. *IBM Systems J. 9*, 3 (1970).
13. Popek, G.J., and Kline, C. Verifiable secure operating system software. Proc. NCC 1974, AFIPS Press, Montvale, N.J., pp. 145–151.

Formal Virtualization Reqs.

- 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 **privileged** iff for any pair of states $S_1 = \langle e, super, p, r \rangle$ and $S_2 = \langle e, user, p, r \rangle$ in which $i(S_1)$ and $i(S_2)$ do not memory trap: $i(S_2)$ traps and $i(S_1)$ does not.
- Example: … many
- Def: Instruction i is **control sensitive** if there exists a state $S_1 = \langle e_1, m_1, p_1, r_1 \rangle$, and $i(S_1) = S_2 = \langle e_2, m_2, p_2, r_2 \rangle$ such that $i(S_1)$ does **not** memory trap, and either $r_1 \neq r_2$, or $m_1 \neq m_2$, or both.
- Example: manipulate status register, return to user mode, etc.

Formal Requirements for Virtualizable Third Generation Architectures

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

Virtual machine systems have been implemented on a limited number of third generation computer systems, e.g. CP-67 on the IBM 360/67. From previous empirical studies, it is known that certain third generation computer systems, e.g. the DEC PDP-10, cannot support a virtual machine system. In this paper, model of a third-generation-like computer system is developed. Formal techniques are used to derive precise sufficient conditions to test whether such an architecture can support virtual machines.

Key Words and Phrases: operating system, third generation architecture, sensitive instruction, formal requirements, abstract model, proof, virtual machine, virtual memory, hypervisor, virtual machine monitor

CR Categories: 4.32, 4.35, 5.21, 5.22

Copyright © 1974, Association for Computing Machinery, Inc. General permission to republish, but not for profit, all or part of this material is granted provided that ACM's copyright notice is given and that reference is made to the publication, to its date of issue, and to the fact that reprinting privileges were granted by permission of the Association for Computing Machinery.

This is a revised version of a paper presented at the Fourth ACM Symposium on Operating Systems Principles, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, October 15-17, 1973.

This research was supported in part by the U.S. Atomic Energy Commission, Contract No. AT(11-1) Gen 10, Project 14 and in part by the Electronic Systems Division, U.S. Air Force, Hanscom Field, Bedford, Massachusetts under Contract Number F19628-70-0217.

Authors' addresses: Gerald J. Popek, Computer Science Department, University of California, Los Angeles CA 90024; Robert P. Goldberg, Honeywell Information Systems, Waltham, MA 02154.

Communications of the ACM
July 1974
Volume 17
Number 7

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) $S_1 = \langle e \mid r, m_1, p, r \rangle$, and
  (b) $S_2 = \langle e \mid r * x, m_2, p, r * x \rangle$,
  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 Requirements for Virtualizable Third Generation Architectures

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

Virtual machine systems have been implemented on a limited number of third generation computer systems, e.g. CP-67 on the IBM 360/67. From previous empirical studies, it is known that certain third generation computer systems, e.g. the DEC PDP-10, cannot support a virtual machine system. In this paper, model of a third-generation-like computer system is developed. Formal techniques are used to derive precise sufficient conditions to test whether such an architecture can support virtual machines.

Key Words and Phrases: operating system, third generation architecture, sensitive instruction, formal requirements, abstract model, proof, virtual machine, virtual memory, hypervisor, virtual machine monitor

CR Categories: 4.32, 4.35, 5.21, 5.22

Copyright © 1974, Association for Computing Machinery, Inc. General permission to republish, but not for profit, all or part of this material is granted provided that ACM's copyright notice is given and that reference is made to the publication, to its date of issue, and to the fact that reprinting privileges were granted by permission of the Association for Computing Machinery.

This is a revised version of a paper presented at the Fourth ACM Symposium on Operating Systems Principles, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, October 15-17, 1973.

This research was supported in part by the U.S. Atomic Energy Commission, Contract No. AT(11-1) Gen 10, Project 14 and in part by the Electronic Systems Division, U.S. Air Force, Hanscom Field, Bedford, Massachusetts under Contract Number F19628-70-0217.

Authors' addresses: Gerald J. Popek, Computer Science Department, University of California, Los Angeles CA 90024; Robert P. Goldberg, Honeywell Information Systems, Waltham, MA 02154.

Communications of the ACM
July 1974
Volume 17
Number 7

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 Requirements for Virtualizable Third Generation Architectures

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

Virtual machine systems have been implemented on a limited number of third generation computer systems, e.g. CP-67 on the IBM 360/67. From previous empirical studies, it is known that certain third generation computer systems, e.g. the DEC PDP-10, cannot support a virtual machine system. In this paper, model of a third-generation-like computer system is developed. Formal techniques are used to derive precise sufficient conditions to test whether such an architecture can support virtual machines.

Key Words and Phrases: operating system, third generation architecture, sensitive instruction, formal requirements, abstract model, proof, virtual machine, virtual memory, hypervisor, virtual machine monitor
CR Categories: 4.32, 4.35, 5.21, 5.22

Copyright © 1974, Association for Computing Machinery, Inc. General permission to republish, but not for profit, all or part of this material is granted provided that ACM's copyright notice is given and that reference is made to the publication, to its date of issue, and to the fact that reprinting privileges were granted by permission of the Association for Computing Machinery.

This is a revised version of a paper presented at the Fourth ACM Symposium on Operating Systems Principles, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, October 15–17, 1973.
This research was supported in part by the U.S. Atomic Energy Commission, Contract No. AT(11-1) Gen 10, Project 14 and in part by the Electronic Systems Division, U.S. Air Force, Hanscom Field, Bedford, Massachusetts under Contract Number F19628-70-0217.

Authors' addresses: Gerald J. Popek, Computer Science Department, University of California, Los Angeles CA 90024; Robert P. Goldberg, Honeywell Information Systems, Waltham, MA 02154.

Communications of the ACM
July 1974
Volume 17
Number 7

Formal Virtualization Reqs. (4)

- “Hybrid” Virtualization (with interpreted instr’s):
- Def: Machine State: S = <E, M, P, R>
  - 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 = <E, user, P, R> 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.

Formal Requirements for Virtualizable Third Generation Architectures

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

Virtual machine systems have been implemented on a limited number of third generation computer systems, e.g. CP-67 on the IBM 360/67. From previous empirical studies, it is known that certain third generation computer systems, e.g. the DEC PDP-10, cannot support a virtual machine system. In this paper, model of a third-generation-like computer system is developed. Formal techniques are used to derive precise sufficient conditions to test whether such an architecture can support virtual machines.
Key Words and Phrases: operating system, third generation architecture, sensitive instruction, formal requirements, abstract model, proof, virtual machine, virtual memory, hypervisor, virtual machine monitor
CR Categories: 4.32, 4.35, 5.21, 5.22

Copyright © 1974, Association for Computing Machinery, Inc. General permission to republish, but not for profit, all or part of this material is granted provided that ACM's copyright notice is given and that reference is made to the publication, to its date of issue, and to the fact that reprinting privileges were granted by permission of the Association for Computing Machinery.
This is a revised version of a paper presented at the Fourth ACM Symposium on Operating Systems Principles, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, October 15–17, 1973.
This research was supported in part by the U.S. Atomic Energy Commission, Contract No. AT(11-1) Gen 10, Project 14 and in part by the Electronic Systems Division, U.S. Air Force, Hanscom Field, Bedford, Massachusetts under Contract Number F19628-70-0217.
Authors' addresses: Gerald J. Popek, Computer Science Department, University of California, Los Angeles CA 90024; Robert P. Goldberg, Honeywell Information Systems, Waltham, MA 02154.

Communications of the ACM
July 1974
Volume 17
Number 7

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.

Unvirtualizable Architectures

UNSW

- x86: lots of unvirtualizable features
  - e.g. sensitive PUSH of PSW is not privileged
  - segment and interrupt descriptor tables in virtual memory
  - segment description expose privileged level
- Itanium: mostly virtualizable, but
  - interrupt vector table in virtual memory
  - THASH instruction exposes hardware page tables address
- MIPS: mostly virtualizable, but
  - kernel registers k0, k1 (needed to save/restore state) user-accessible
  - performance issue with virtualizing KSEG addresses
- ARM: mostly virtualizable, but
  - some instructions undefined in user mode (banked registers, CPSR)
  - PC is a GPR, exception return in MOVS to PC, doesn't trap
- Most others have problems too
- Recent architecture extensions provide virtualization support hacks

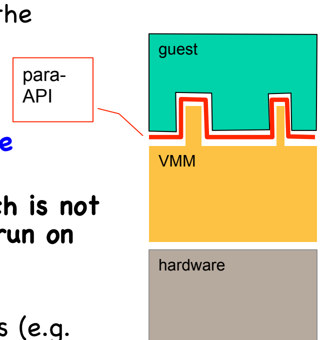
©2008 Gernot Heiser UNSW/NICTA/OKL. Distributed under Creative Commons Attribution License 16

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. XenWindowsGplPv 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 quest 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

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

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

<- small example, C code

same code, compiled ->

```
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       ; a % i
          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
```

Translation: Mechanics

instruction stream

```
89 f9 be 02 00 00 00 39 ce 7d
```

Translation Unit (TU)

```
isPrime:  mov %ecx, %edi
          mov %esi, $2
          cmp %esi, %ecx
          jge prime
```

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)

```
isPrime':   mov %ecx, %edi    ; IDENT
            mov %esi, $2
            cmp %esi, %ecx
            jge [takenAddr]   ; JCC
            jmp [fallthrAddr]
```

Translation Result

```
isPrime:   mov    %ecx, %edi  ; %ecx = %edi (a)
           mov
           cmp
           jge
nexti:     mov
           cdq
           idiv
           test
           jz
           inc
           cmp
           jl
prime:     mov
           ret
notPrime:  xor
           ret
```

```
isPrime':  *mov   %ecx, %edi    ; IDENT
            mov   %esi, $2
            cmp   %esi, %ecx
            jge   [takenAddr]   ; JCC
                                ; fall-thru into next CCF
nexti':    *mov   %eax, %ecx    ; IDENT
            cdq
            idiv  %esi
            test  %edx, %edx
            jz    notPrime'     ; JCC
                                ; fall-thru into next CCF
           *inc   %esi          ; IDENT
            cmp   %esi, %ecx
            jl    nexti'        ; JCC
            jmp   [fallthrAddr3]

notPrime': *xor   %eax, %eax    ; IDENT
            pop   %r11          ; RET
            mov   %gs:0xff39eb8(%rip), %rcx   ; spill %rcx
            movzx %ecx, %r11b
            jmp   %gs:0xfc7dde0(8*%rcx)
```

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.

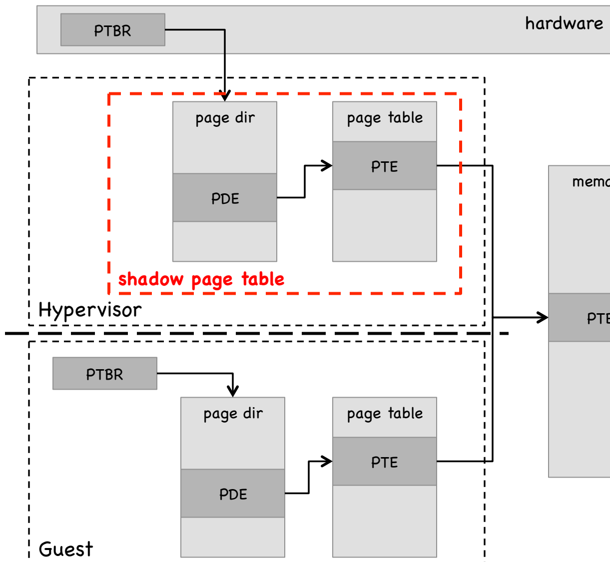
Memory Virtualization

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

- In traditional system:
  virtual address -> physical address
- In VMM system:
  virtual address -> physical address -> **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 -> machine mapping.

Shadow Page Table



Hypervisor maintains mapping from virtual memory to machine memory in **shadow page table**.

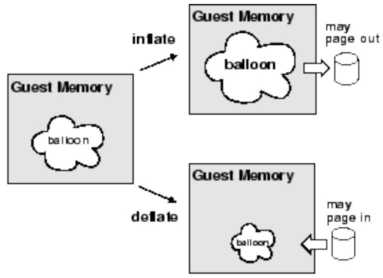
Guest modifies its page mapping, either by changing the content of a translation, creating a new translation, or removing an existing translation.

=> The virtual MMU module captures modification and adjusts the **shadow page table** 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?

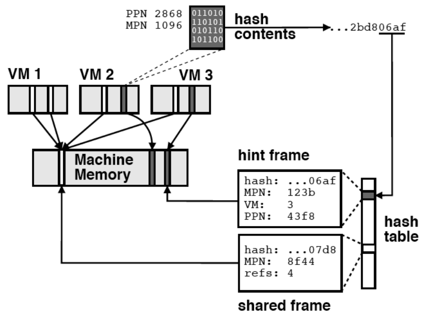
How to Adjust Memory Allocation

- 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.*

Light-Weight "Virtualization": Containers

Container: A group of processes that is grouped together and isolated from processes in other containers.

"Insider the box, it looks like a VM.
Outside the box, it looks like normal processes."

Container: Advantages

- **Speed**: "boots" in seconds, i.e. much faster than VM
- **Footprint**: can run order-of-magnitude more containers than VMs.
- **Memory footprint**: containers can be very light.
- **Isolation**
  - more about this follows.

Container Isolation

Each container has:

- its own network interface (and IP address)
- its own filesystem
- isolation (security)
  - container A cannot harm (or even see) container B.
- isolation (resources)
  - soft and hard quotas

Isolation: Namespaces

6 different kinds of namespaces:

- Process ids (pid)
- Network interfaces (net)
- System V IPC (ipc)
- File systems and mount points (mnt)
- Hostname (uts)
- User IDs (user)

Example: Namespace pid

- Requirement: Processes in a pid namespace don't see processes in another pid.
- Requirement: Each pid namespace has a PID #1

Container: The Buzz

Containerization is the new virtualization

Containers are in use by many PaaS (Platform as a Service) companies; to mention a few -

- **dotCloud** (which changed later its name to docker): https://www.dotcloud.com/
- **Parallels** - http://www.parallels.com
- **Heroku** - https://www.heroku.com/
- **Pantheon** - https://www.getpantheon.com/
- **OpenShift** of Red Hat: https://www.openshift.com/
- more.

http://ramirose.wix.com/ramirosen 31