Hardware-basierte Virtualisierung

Dr.-Ing. Volkmar Sieh

Department Informatik 4
Verteilte Systeme und Betriebssysteme
Friedrich-Alexander-Universität Erlangen-Nürnberg

WS 2021/2022

Hardware-basierte Virtualisierung

Idee:

- Host-CPU kann sich selbst perfekt emulieren.
- Host-CPU emuliert sich selbst in Echtzeit.
- Nehmen wir doch die...

Bedingung (große Einschränkung!):

- Zu emulierende CPU = Host-CPU.

Wichtig/schwierig:

- Sicherstellen, dass eine laufende VM nicht das Host-OS, andere Applikationen oder andere VMs stören kann.
- Kritische Instruktionen „ausfiltern” und auf bekannte Weise emulieren.

Hardware-basierte Virtualisierung

Idee nicht neu (z.B. Paper von Popek und Goldberg, 1974, IBM-Mainframes).

Neu: 80x86-Erweiterungen, um kritische Instruktionen auszufiltern

AMD: Pacifica

Intel: Vanderpool (VT-x)

Hardware-basierte Virtualisierung – „Kritische" Befehle

Unterscheidung von Befehlen; Beispiele 80x86-Befehle:

`cli`, `sti`, `inb`, `outb`, `mov *, %cr*`, `...`: Instruktionen, die im User-Mode der CPU nicht ausgeführt werden können („Privilege Violation"-Exception). Problemlos.

`popf`: Instruktionen, die sich im User-Mode und Supervisor-Mode der CPU anders verhalten. **Problematisch!**

`sidt`, `sgdt`, `...`: nicht-priviligierte Instruktionen, die Informationen abfragen, die emuliert werden sollen. **Problematisch!**

übrige Befehle: Problemlos.

Hardware-basierte Virtualisierung

Beispiel (Auslesen eines im Interrupt beschriebenen Puffers):

```
  pushf
  cli                  <- kritisch
  cmpl $0, avail
  je .L1
  movzbl buf, %eax
  movl $0, avail
  jmp .L2
.L1:
  movl $-1, %eax
.L2:
  popf                 <- kritisch
  ret
```

`cli` generiert im User-Mode Exception. Mit aktiviertem Virtualisierungs-Modus unter Pacifica/Vanderpool auch `popf`.

Hardware-basierte Virtualisierung

Verschiedene Typen kritischer Instruktionen:

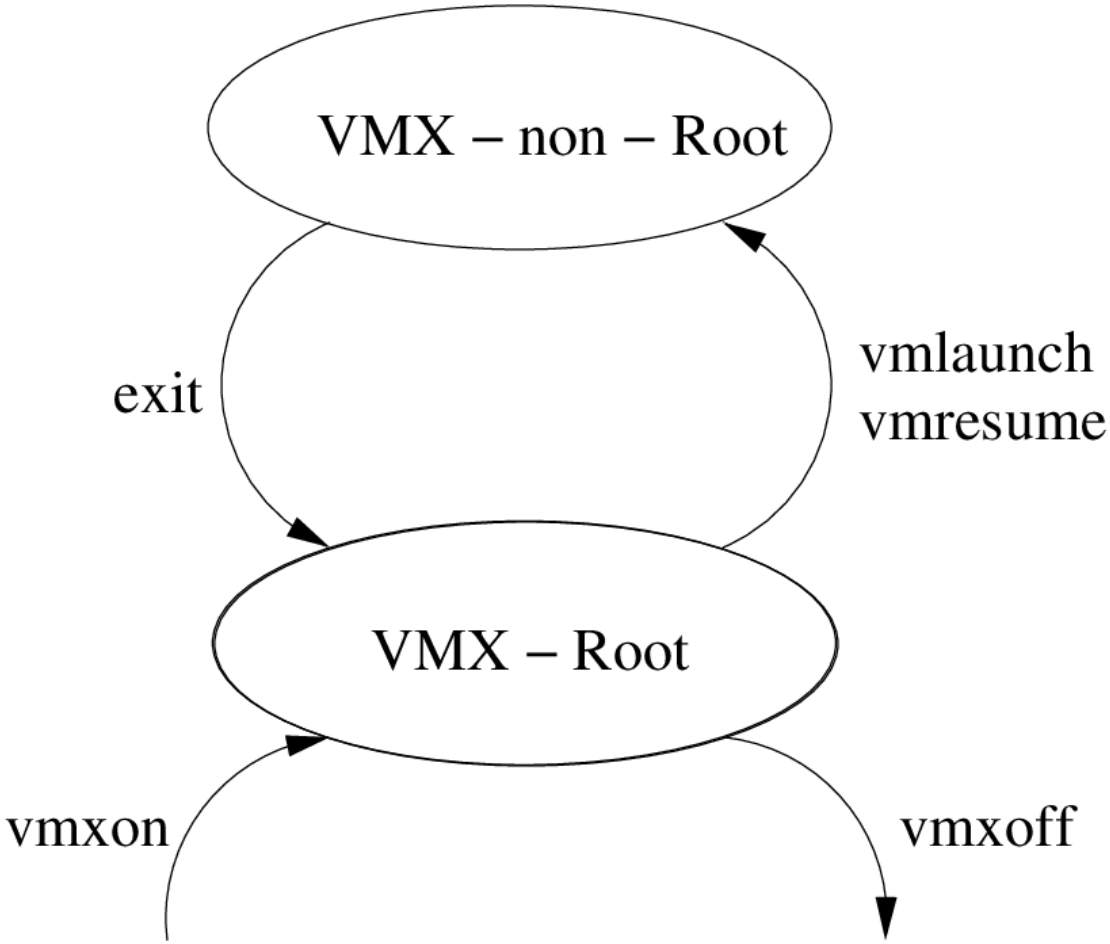
`inb`, `outb`: Ein-/Ausgabe soll i.A. emuliert werden.

`cli`, `sti`: Sollen *virtuelle* Interrupts disablen/enablen.

`mov *, %cr*`: Soll eigentlich so ablaufen (Voraussetzung: übertragener Wert „zulässig").

`...`:

Hardware-basierte Virtualisierung



Zustände der Intel-VMX-Erweiterung

Hardware-basierte Virtualisierung

- `vmlaunch`, `vmresume` starten CPU-Emulation bzw. setzen sie fort. Die Assembler-Instruktionen speichern den aktuellen CPU-Zustand ab und laden den Zustand der zu emulierenden CPU.
- Ein `exit`-Event tritt auf, wenn z.B.
  - `in`- oder `out`-Instruktionen ausgeführt werden sollen,
  - eine `hlt`-Instruktion ausgeführt werden soll,
  - Control-Register beschrieben werden sollen,
  - ein Interrupt auftritt,
  - ...

  Dann wird der aktuelle Zustand der zu emulierenden CPU abgespeichert und der beim `vmlaunch`/`vmresume` gespeicherte Host-CPU-Zustand zurückgeladen.

Hardware-basierte Virtualisierung – I/O

Zugriffe auf I/O-Geräte müssen i.A. immer virtualisiert werden. Sonst kann eine virtuelle Maschine den Host stören.

- `inb`, `outb`, ... müssen *immer* abgefangen werden („Privilege Violation").
- `movb`, `movw`, ... müssen *–wenn sie auf I/O-Geräte gehen–* abgefangen werden („Page Fault").

Virtualisierung hier kein Problem. Entsprechende Exceptions existieren bei allen relevanten CPUs.

U.U. problematisch: Umschaltzeiten zur/von der Geräte-Emulation.

Hardware-basierte Virtualisierung – I/O

Ungünstig: I/O über viele einzelne `in`-/`out`-Instruktionen;
Beispiele: viele alte I/O-Geräte (IDE, NE2000, ...)

Günstig: I/O über wenige `in`-/`out`-Instruktionen und (virtuelles) DMA sowie Command-/Buffer-Chaining;
Beispiele: viele neue I/O-Geräte (EIDE, EEPRO100, ...)

Und Grafik...?

Hardware-basierte Virtualisierung – I/O (Grafik)

Zugriff auf Grafik-Karte über *sehr* viele `mov`-Instruktionen.

=> sehr viele Wechsel virtueller Modus <=> normaler Modus

=> katastrophale Performance...!

Hardware-basierte Virtualisierung – I/O (Grafik)

Drei Möglichkeiten:

- `mov`-Instruktionen auf Funktionsaufrufe abbilden
  - katastrophale Performance bei Grafik-Operationen
  - optimale Performance bei sich nicht änderndem Bildschirminhalt
  - Nachbildung von GPUs möglich
- `mov`-Instruktionen einfach in den Grafik-Speicher schreiben lassen (ohne Bildschirm-Update); Bildschirm-Update periodisch durch zweiten Prozess (FAUmachine: 5mal pro Sekunde)
  - perfekte Performance bei Grafik-`mov`-Operationen
  - dauernde Belastung durch Bildschirm-Update (auch wenn keine/kaum Schreiboperationen durchgeführt wurden; FAUmachine: ca. 10% der Rechenleistung bei 1024x768)
  - keine Nachbildung von GPUs möglich
- Für diesen Zweck entwickelte Grafik-„Hardware" mit speziellem Grafik-Treiber verwenden.

Hardware-basierte Virtualisierung – Interrupts

Interrupts dürfen vom Guest nicht abgeschaltet werden können. Sonst kann eine virtuelle Maschine den Host stören.

`cli`, `sti` und `popf` können unter Vanderpool/Pacifica Exceptions generieren.

Problem: `cli`/`popf`-Aufrufe sehr häufig; Umschaltzeiten groß.

Lösung: Vanderpool/Pacifica merken sich das Interrupt-Enable-Bit der virtuellen Maschine, ändern aber das Interrupt-Enable-Bit des Hosts nicht. Interrupts des Hosts gehen an das Host-OS.

Hardware-basierte Virtualisierung – Interrupts

Wenn Interrupts an den Host gehen – wie bekommt der Guest Interrupts?

Beim `vmresume` können dem Guest „Events” (Interrupts) mitgegeben werden.

Hardware-basierte Virtualisierung – Exceptions

Exceptions oder System-Calls in der virtuellen Maschine können in der virtuellen Maschine selbst abgehandelt werden.

Vanderpool/Pacifica haben für eine virtuelle Maschine ein Extra-Register zum Speichern der Adresse der Interrupt/Exception/System-Call-Descriptor-Tabelle.

`lidt`-/`sidt`-Instruktionen funktionieren daher in der virtuellen Maschine ganz normal, ändern aber ein anderes Register als im Host.

Hardware-basierte Virtualisierung – Segmentierung

Segmentierung hat keinen Einfluss auf die Sicherheit des Host-OS. Sicherheit bei Speicherzugriffen wird bei modernen Systemen nur noch über Paging hergestellt.

Das Laden/Speichern der Segment-Register (`mov ..., %?s`, `mov %?s, ...` und der Segment-Descriptor-Tabellen-Pointer (`lgdt`, `lldt`, `sgdt`, `sldt`) kann daher von der Hardware zugelassen werden.

Hardware-basierte Virtualisierung – Segmentierung

Ausnahme:

- Bei x86-Prozessoren enthält Code-Segment-Register den Privilege-Level.

Lösung: Es gibt unter Vanderpool/Pacifica 4 zusätzliche Privilege-Level:

- Host
  - Ring 0: höchste Priviligierungsstufe (i.A. Host-OS)
  - ...
  - Ring 3: niedrigste PS (i.A. Host-User-Level)
- Guest
  - Ring 0: höchste Priviligierungsstufe (i.A. Guest-OS)
  - ...
  - Ring 3: niedrigste PS (i.A. Guest-User-Level)

Hardware-basierte Virtualisierung – MMU

- `mov ..., %cr3` (Page-Directory-Basis-Register laden) ist die kritische Instruktion (gebraucht bei jedem Kontext-Wechsel).
- `mov ..., %cr0` und `mov ..., %cr4` kritisch, aber nur selten gebraucht.
- `invlpg` unkritisch.

Hardware-basierte Virtualisierung – MMU

Laden des Page-Directory-Basis-Registers ist kritisch für die Sicherheit des Hosts und kritisch für die Performance des Gastes:

Hardware stellt 4 (Intel) Werte zur Verfügung. Wenn einer der 4 Werte dem Wert entspricht, der in das `%cr3`-Register geladen werden soll, wird (ohne Exception) geladen. Sonst wird virtuelle Maschine unterbrochen.

$=>$ schnelle Kontext-Wechsel, wenn wenige Prozesse lauffähig sind.

Check und ggf. Umbau der Page-Tabellen nur beim erstmaligen Start einer Applikation.

Hardware-basierte Virtualisierung

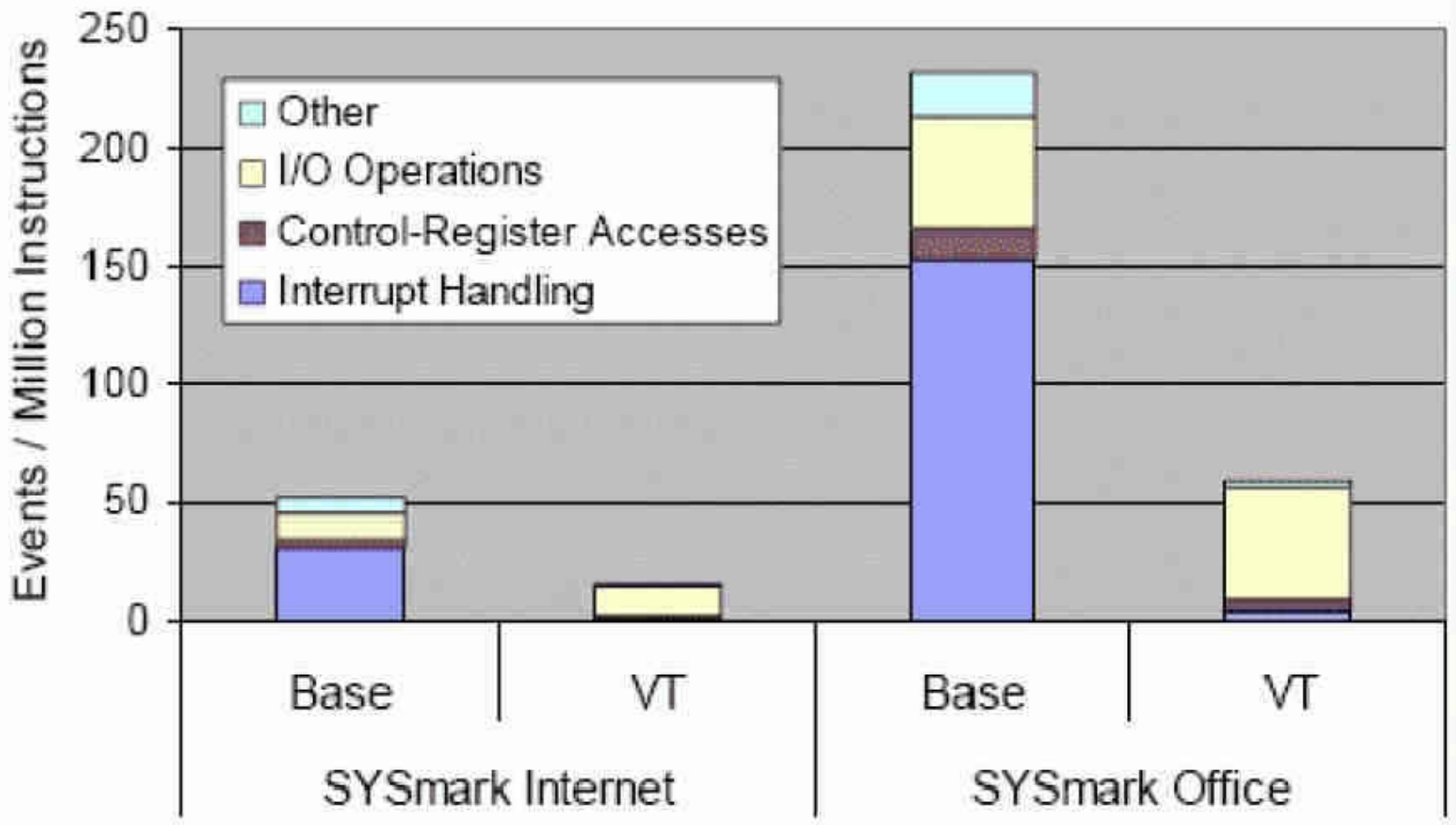
Hardware-basierte Virtualisierung (hier: Pacifica/Vanderpool) erlauben

- I/O-Geräte zu virtualisieren
  - mit Performance-Einbußen
- die CPU direkt zu verwenden

  MMU: mit Performance-Einbußen

  Rest: ohne Performance-Einbußen

Hardware-basierte Virtualisierung



Quelle: Diplomarbeit T. Lindinger

Hardware-basierte Virtualisierung – User-Mode

Und ohne teure/neue CPU...?

Im Allgemeinen problemlos:

Emulation der CPU durch sich selbst im **User-Modus**.

Idee:

- Standard-Emulation, solange VM im Supervisor-Modus.
- Nutzung der Host-CPU, wenn VM im User-Modus.

Umschaltung sehr aufwändig:

- Übertragung der emulierten in die realen Register
- Umprogrammierung der Segmentierung
- Umprogrammierung der MMU; Aufbau neuer Page-Tabellen

Lauf der realen CPU muss durch reale Interrupts unterbrechbar sein
=> nicht deterministisch.