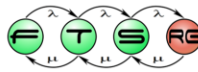


CPU virtualizáció

Micskei Zoltán, Tóth Dániel



Utolsó módosítás: 2011. 09. 22.

Mi történt az elmúlt két hétben?

Release: VMware Workstation 8.0

Posted by Kenneth van Surksom | Wednesday, September 14th, 2011 | 0 Comments and 13 Readings



VMware has **released** version 8 of its virtualization platform Workstation. While normally in-line with Workstation also Player and ACE are updated, no information on an updated version for these two products can be found (yet).

This version of Workstation includes the following new features:

Microsoft provides more details about Windows Server 8

Posted by Kenneth van Surksom | Wednesday, September 14th, 2011 | 0 Comments and 45 Readings



During its annual **Build conference**, Microsoft shared some information about the upcoming versions of its flagship product Windows. As virtualization.info already **reported** the client version of Windows, dubbed Windows 8 will contain the Hyper-V hypervisor technology, besides other new functionality.



Bringing Hyper-V to "Windows 8"

14 days ago by [Steven Sinofsky](#)

★★★★★
177

In this post we talk about how we will support virtualization on the Windows "client" OS. Originally released for Windows Server where the technology has proven very popular and successful, we wanted to bring virtualization to a core set of...

VMware announces Horizon Mobile

Posted by Kenneth van Surksom | Wednesday, September 7th, 2011 | 0 Comments and 11 Readings



In November 2008, VMware **acquired** Trango Virtual Processors announcing its plans to release a type-2 hypervisor for mobile phones called Mobile Virtualization Platform (MVP). Development didn't go as planned though, forcing VMware to **delay the solution for at least one year**. In December last year, VMware **announced** a partnership with LG to deliver MVP on LG phones with the Android OS.



Források:

- <http://virtualization.info/en/>
- <http://blogs.msdn.com/b/b8/>
- <http://blogs.vmware.com/console/>

Tartalom

Platform virtualizáció technikai háttere

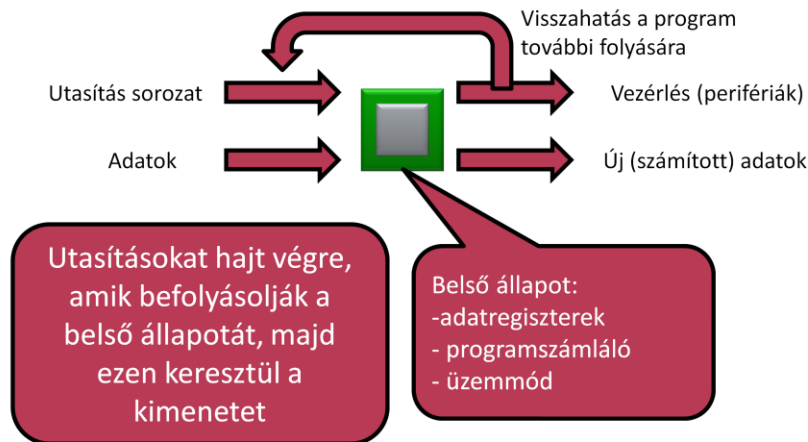
- **Ismétlés: mit csinál egy CPU**
- Mi kell egy virtuális CPU-hoz (Popek & Goldberg)
- Szimuláció, emuláció, virtualizáció
- Három lehetőség a virtualizációra: szoftveres, hardveres, para-
- Memóriakezelés egy modern CPU-ban
- Virtuális gépek memóriakezelése: szoftveres, hardveres, para-
- Virtuális memóriakezelés speciális képességei: megosztás, késleltetett allokáció, memória-ballon



Mi is az a processzor?

Egyszerű matematikai modellje:

- Belső állapottal rendelkező automata



Kicsit konkrétan:

- az utasítás hatására a belső regiszterek valamelyikének értékét módosítja, felhasználva regiszter értékeket és/vagy kívülről betöltött adatot.
- A belső regiszter tartalma lehet később kimenő adat,
- vagy befolyásolhatja a program további futását, pl. a programszámlálót átírja, ezzel ugrik más programrészhez (erre általában dedikált vezérlési-folyam utasítások vannak: jump/goto, branch/if, call, return).

Mi is az a processzor?

CPU utasításkészlet architektúrája (ISA, Instruction Set Architecture)

- Meghatározza a lehetséges **utasításokat**
 - bináris kódjait (opcodes) és assembly rövidítéseit (mnemonic)
 - operandusok fajtáját (konstans, regiszter, memória, indirekt memória)
 - végrehajtási szemantikájukat: pontos jelentésüket, hatásukat
- Meghatározza a **regiszterkészletet**
 - általános célú adattároló regiszterek, méretükkel és a bennük tárolható adat formátumával (előjeles/nélküli egész, lebegőpontos, logikai stb.)
 - speciális regiszterek, pl. programszámláló, verem mutató...
- CPU **üzemmódjait**
 - Minden üzemmódban más-más részhalmaza engedélyezett az utasításkészletnek
 - Memória elérése lényegesen eltérhet üzemmódok között

ISA – példa

- Részlet az Intel x86 kézikönyvből:

INSTRUCTION SET REFERENCE, A-M

CPUID—CPU Identification

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
0F A2	CPUID	Valid	Valid	Returns processor identification and feature information to the EAX, EBX, ECX, and EDX registers, as determined by input entered in EAX (in some cases, ECX as well).

Description

The ID flag (bit 21) in the EFLAGS register indicates support for the CPUID instruction. If a software procedure can set and clear this flag, the processor executing the procedure supports the CPUID instruction. This instruction operates the same in non-64-bit modes and 64-bit mode.



Intel® 64 and IA-32 Architectures Software Developer Manuals

<http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html/>

AMD Developer Guides & Manuals

<http://developer.amd.com/documentation/guides/pages/default.aspx#manuals>

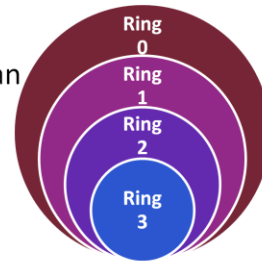
Mit tud a processzor – üzemmódok

■ A CPU üzemmódok célja:

- Visszamenőleges kompatibilitás (x86 másból sem áll...)
 - Pl.: valós mód (16 bit), védett mód (16 és 32 bit), long mód (64 bit)...
- Egy program ne tudjon bizonyos műveleteket végezni
- Operációs rendszer el tudja szigetelni a programokat egymástól
 - „védett” módok
 - Futási **privilegium szintek**, al-üzemmódok (rings)

■ Példa: 4 privilegium szint az x86 védett módjában

- Ring 0, *supervisor* mód: legbővebb utasításhalmaz
- Ring 1
- Ring 2
- Ring 3, *user* mód: legszűkebb részhalmaz



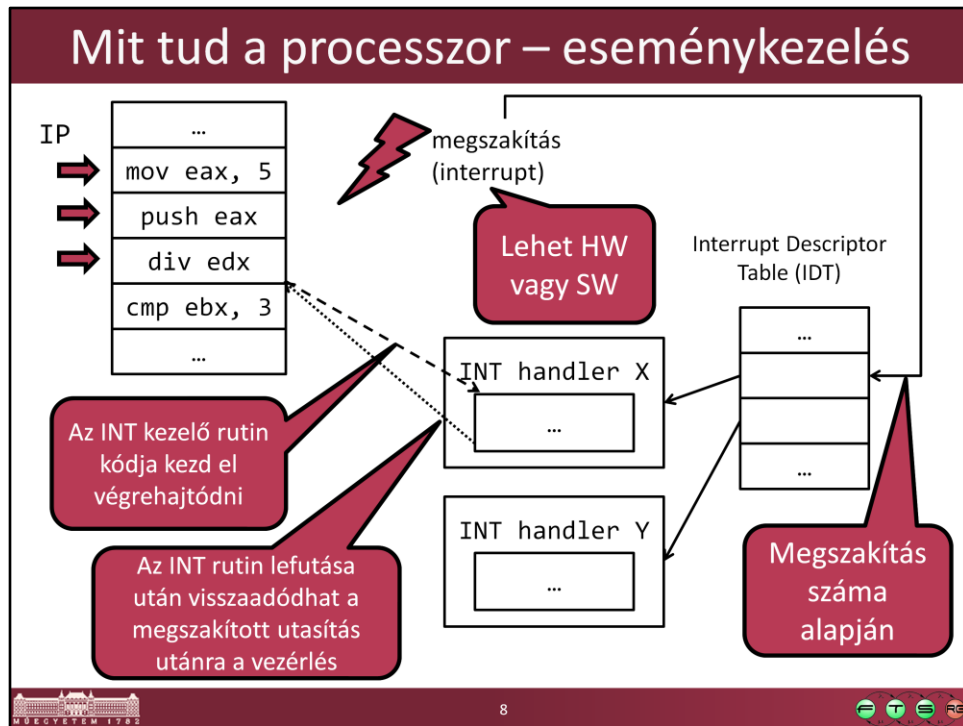
Az ISA specifikáció biztosítja, hogy egy program - azonos bemenő adattal ellátva - minden esetben ugyanúgy fusson le, ugyanazt az eredmény adja.

Elszigetelés:

- Az elszigetelés fő célja, hogy az egyik program hibája esetén egy másik futó program zavartalanul működhessen tovább ugyanazon a processzoron.
(Megjegyzés: az ábrát gyakran fordítva szokták rajzolni, hogy a Ring0 van legbelül. Most ez halmazokat hivatott illusztrálni, ezért van a Ring 0, mint legbővebb utasításhalmaz, kívül.)
- Mit jelent a Ring x86-on: a CPU-nak van egy aktuális futási szintje (CPL – current privilege level) és vannak különböző adatszerkezet leírók szegmens és lapozott memóriához, melyek szintén tartalmazznak egy futási szintet (DPL – descriptor privilege level). Hozzáférés (olvasás, írás, végrehajtás) általános esetben akkor engedélyezett, ha a $CPL \leq DPL$, a bonyolultabb esetekre, mint Call Gate használata most nem térünk ki. A Ring 0-ban a teljes ISA rendelkezésre áll, 1-3-ban csak egy szűkített részhalmaz (pl. védelmi szint, szegmens leíró tábla regiszterek nem válthatók).

Részletes leírás: Intel Architecture Software Developer's Manual 4-9 fejezet.

<http://download.intel.com/design/PentiumII/manuals/24319202.pdf>



(Az ábra egyszerűsít, a lényeget próbálja kiemelni)

Megszakítás (interrupt) beérkezése állapotváltással jár

1. Befejezi az éppen aktuális utasítást
2. A megszakítás száma (csatornája) alapján a CPU egy meghatározott címen folytatja a végrehajtást
3. Lehet hardveres (perifériától származó) és szoftveres eredetű (INT utasítás x86-on)
4. Interrupt descriptor table – egy tömb ami tartalmazza a megszakítás számához tartozó kezelő (handler) programrészletre mutató pointert
5. Megszakítás válthatja a CPU üzemmódját is (felhasználói módból máshogy nem is lehet kikerülni)
6. A szoftveres megszakításból alakultak ki a modernebb rendszerhívás (syscall) utasítások

Mit tud a processzor – eseménykezelés

OS alapvető része a megszakításkezelés

- Így kezeli a **perifériáktól** érkező eseményeket
- Az **időzítő periféria** periodikus megszakításküldésével jut vissza a vezérlés rendszeresen a rendszermaghoz (OS ütemező)
- Az alkalmazások szoftveres megszakítással vagy rendszerhívással kérhetik meg a rendszermagot különböző műveletekre

Mit tud a processzor – eseménykezelés

Kivételek (exception):

- Ha a programban olyan utasítás kerül sorra, ami az adott üzemmódban nem engedélyezett, akkor kivétel keletkezik (Illegal Instruction *Exception* vagy *Trap*)
- Ilyenkor is váltás történik a supervisor módba és a vezérlés egy megadott memóriaterületen lévő kezelő rutinhoz kerül
- A rendszermag mindig értesül a user módban futó folyamatok „tilosban járásairól” is
- (Vannak további fajta kivételek, pl. page fault...)



10



Pl. x86-on előre definiált kivételek:

- 0 – divide error (#DE),
- 4 – overflow (#OF),
- 14 – page fault (#PF),
- 32-255 – user defined
- ...

--

Az x86 architektúrán az elnevezés kicsit összetettebb, három fajta kivétel van:

- Fault: a kivétel kezelése után a kivételt okozó utasítás elé térünk vissza, így azt meg lehet próbálni újra végrehajtani
- Trap: a kivétel kezelése után a kivételt okozó utasítás utánra térünk vissza
- Abort: súlyos hiba, nem térünk vissza a megszakított feladatra

Tartalom

- Platform virtualizáció technikai háttere
 - Ismétlés: mit csinál egy CPU
 - **Mi kell egy virtuális CPU-hoz (Popek & Goldberg)**
 - Szimuláció, emuláció, virtualizáció
 - Három lehetőség a virtualizációra: szoftveres, hardveres, para-
 - Memóriakezelés egy modern CPU-ban
 - Virtuális gépek memóriakezelése: szoftveres, hardveres, para-
 - Virtuális memóriakezelés speciális képességei: megosztás, késleltetett allokáció, memória-ballon

Virtuális CPU feltételei

- Popek és Goldberg* követelményei egy virtualizált rendszerrel szemben
 - VMM – *virtual machine monitor*: olyan komponens, ami a virtuális gépek számára az absztrakciót biztosítja
- **Ekvivalencia**
 - A VMM felett futó program mindig pontosan ugyanazt az eredményt adja futás közben, mintha fizikai CPU-n közvetlenül futna
- **Erőforrás kezelés**
 - A VMM minden virtualizált erőforrást teljes egészében maga felügyel
- **Hatékonyág**
 - A virtuális gépben futó program utasításainak nagy része változtatás és VMM beavatkozás nélkül fut a fizikai CPU-n



Kapcsolódó cikk:

Gerald J. Popek and Robert P. Goldberg. 1974. Formal requirements for virtualizable third generation architectures. *Commun. ACM* 17, 7 (July 1974), 412-421.
DOI=10.1145/361011.361073, <http://doi.acm.org/10.1145/361011.361073>

Virtuális CPU feltételei

- Mik a problémák
 - Az OS felett futó folyamat korlátozottabb CPU üzemmódban fut, közvetlenül nem menedzsel hardver erőforrásokat, nem futhat OS nélkül – ez nem ekvivalens futási környezet
 - OS nem futhat user módban, mert hardver erőforrásokat kell közvetlenül menedzselnie, ehhez kellenek a supervisor mód nyújtotta privilégiumok.
- Az ekvivalencia és az erőforrás kezelési kritérium ellentmondó követelményeket támaszt

Tartalom

Platform virtualizáció technikai háttere

- Ismételtes: mit csinál egy CPU
- Mi kell egy virtuális CPU-hoz (Popek & Goldberg)
- **Szimuláció, emuláció, virtualizáció**
- Három lehetőség a virtualizációra: szoftveres, hardveres, para-
- Memóriakezelés egy modern CPU-ban
- Virtuális gépek memóriakezelése: szoftveres, hardveres, para-
- Virtuális memóriakezelés speciális képességei: megosztás, késleltetett allokáció, memória-ballon

Virtuális CPU feltételei

■ Ekvivalencia biztosítása

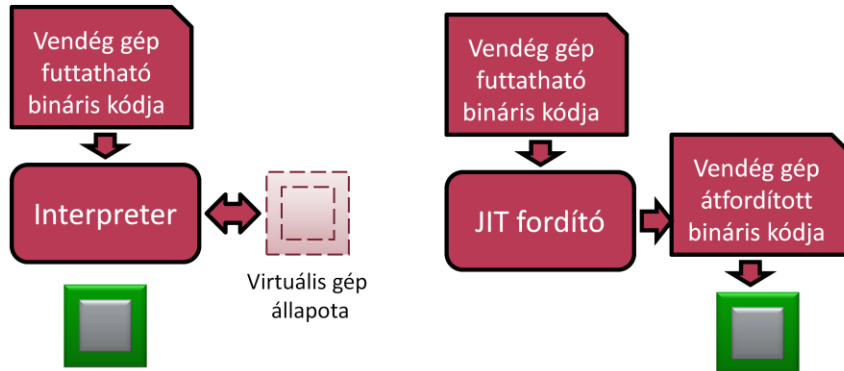
- **Szimuláció:** szoftverrel modellezzük a processzor belső működését (különböző mélységig megtartva a valósághűséget). Fizikailag sohasem fut a virtuális gép kódja a CPU-n.
- **Emuláció:** helyettesítjük a VMM felett futtatandó programot/annak egy részletét egy másikkal, ami végül pontosan ugyanazt az eredményt adja (de akár teljesen más futási utat bejárva, elkerülve a privilegizált utasítások használatát)



Figyelem, az emuláció vs. szimuláció csak ebben a környezetben definiálható így. Más tudományterületek eltérő értelemben is használják.

Tiszta emuláció

- A vendég virtuális gép kódját a processzor nem futtatja közvetlenül, hanem adatként feldolgozza
 - Eltérhet a virtuális gép CPU architektúrája a futtató CPU-tól
 - Virtualizációhoz (P&G értelemben vett) képest lassú



A két ábra kétféle megvalósítási módot mutat.

Virtuális CPU feltételei

Emuláció (és szimuláció) kétféleképpen

- **Futási idejű értelmező (Interpreter)** – adatként kezeli és lépésenként hajtja végre egy szoftveres modellen a virtuális gép utasításfolyamát
 - Lassú, a CPU közvetlenül csak az értelmező kódját futtatja
 - Hordozható futtatókörnyezet
 - Izoláció természetesen adódik
 - Példa: Bochs x86 emulátor
- **„Éppen időben” fordító (JIT compiler „just in time”)** – végrehajtás előtt egy fordító feldolgozza a virtuális gép soron következő utasításait, és kódot generál belőle, ami az eredetivel ekvivalens viselkedést mutat
 - Közvetlenül futtatható kódot generál, cache-elési eljárásokkal gyors lehet
 - Nehéz implementálni, nem hordozható
 - Izolációt nem automatikusan biztosítja
 - Példa: QEMU x86 emulátor (és Java VM, .NET CLR is ilyen)



Interpreter hogyan hordozható: ha pl. C nyelven írjuk (bizonyos kódolási konvenciók betartásával), akkor bármely célplatformon futtatható, amire van C fordító
JIT fordító viszont specifikus kell hogy legyen minden virtuális gép-futtató platform párra.

A Java VM és a .NET CLR is Jit fordítós emulátorok, olyan architektúrát emulálnak, ami fizikai processzoron sohasem létezett. Viszont ezek alkalmazásfuttató környezetek, NEM tartoznak a platform virtualizáció témakörébe!

Tartalom

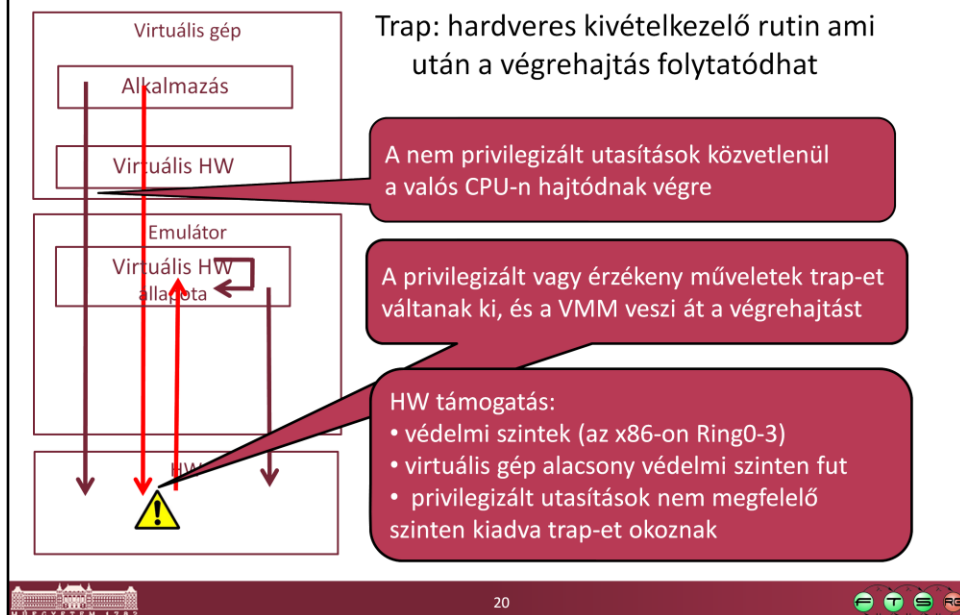
Platform virtualizáció technikai háttere

- Ismételtes: mit csinál egy CPU
- Mi kell egy virtuális CPU-hoz (Popek & Goldberg)
- Szimuláció, emuláció, virtualizáció
- **Három lehetőség a virtualizációra: szoftveres, hardveres, para-**
- Memóriakezelés egy modern CPU-ban
- Virtuális gépek memóriakezelése: szoftveres, hardveres, para-
- Virtuális memóriakezelés speciális képességei: megosztás, késleltetett allokáció, memória-ballon

Három lehetőség a virtualizációra

- **Virtualizáció** – az utasításokat (egy részüket) változatlanul hagyja végrehajtani, csak a problémás privilegizáltakkal kell valamit kezdeni
 - **Szoftveres virtualizáció** (Trap and Emulate + bináris fordítás)
 - **Hardveres virtualizáció** (Trap and Emulate, teljesen hardveres támogatással)
 - **Paravirtualizáció** (vendég OS kódját módosítjuk, hogy tudjon a virtualizációról, és hívhassa a VMM-et)

Elfogás és emuláció elve



Virtuális gép alacsonyabb védelmi szinten futtatása: deprivegizálás (de-privileging)
x86 ring: 4 darab van, 0 a legmagasabb (itt futnak általában az OS-ek kerneljei), 3 a legalacsonyabb (itt futnak az alkalmazások)

A processzor trap művelete kb olyan, mint egy kivétel (exception): van egy „catch” blokkja, ami egy a processzorba fixen beállított lekezelő függvénypointer tömb.
(Megjegyzés az x86-nak valójában van egy 5. ringje is, a System Management Mode, ami a ring 0-nál is bővebb utasításhalmazt enged meg, és el tudja fogni a ring0-ban futó utasításokat is.)

Elfogás és emuláció

- Gyakorlatban akkor alkalmazható, ha a CPU minden privilegizált utasítás elfogását támogatja
 - Az x86 nem ilyen
 - Gyakorlatilag egyik CPU architektúra sem ilyen, amelyiket nem kifejezetten erre terveztek
- Példák:
 - POPF utasítás: EFLAGS regisztert módosítja
 - Ha nem ring 0-n adjuk ki, akkor nem ír felül bizonyos biteket, és nem is dob kivételt
 - Privilegizált állapot kiolvasható
 - CS szegmens regiszter 0. és 1. bitjeiből az aktuális ring kiolvasható, így a vendég gép megtudhatja, hogy virtualizált

OS-ek működése



- Az OS normális működése
- SYSCALL és INT (szoftveres megszakítás) a Ring3-ból Ring0-ba hív át
- Az időzítő megszakítás (timer interrupt) rendszeresen a kernel megfelelő kezelő rutinjának adja a vezérlést (ez hívja az ütemezőt)

Szoftveres virtualizáció

Ring 3 – userspace
alkalmazások

Ring 2

Ring 1 – vendég OS

Ring 0 – host OS

A VMM-ben tett „kör”
költséges, a CPU
üzemmód váltás során
teljes állapotmentés
majd a végén
visszaállítás kell!

- Mikor éppen virtualizált operációs rendszer fut
- SYSCALL és INT továbbra is a Ring3-ból Ring0-ba hív át!
- A VMM megkapja a vezérlést, feladata hogy továbbítsa a hívást a vendég kernelnek
- Amikor a vendég kernel Ring1-ben nem engedélyezett utasítást hajtana végre a VMM elkapja, lekezeli, és úgy tesz, mintha megtörtént volna.
- A VMM kapja a timer interruptot is, így ütemezheti a vendég gépeket. Ezt is továbbítja az aktív vendég kernel felé is.
- A vendég OS – teljesítménytől eltekintve – nem tud róla, hogy virtualizált



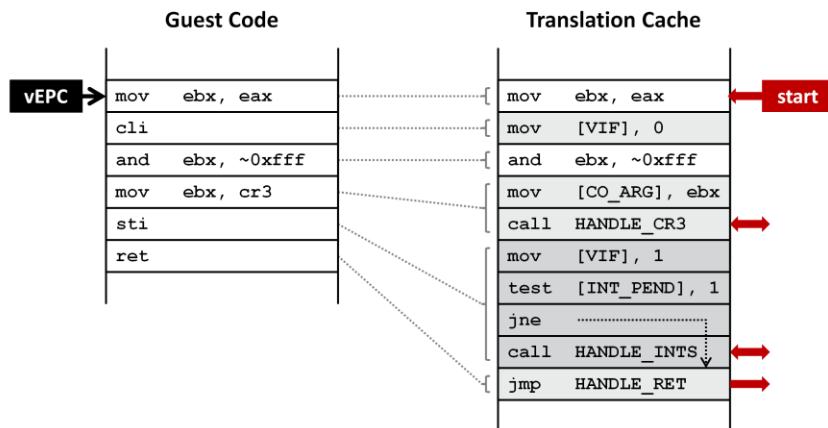
23



Itt látható, hogy baj van, ha van olyan utasítás, ami nem vált ki trap-et, de nem is hajtódik végre Ring3-ban (vagy nem ugyanúgy hajtódik végre), mert akkor nincs esély rá, hogy elkapja a VMM és emulálni tudja a hatását. Így viszont a hatástalan utasítás miatt a vendég OS működése hibás lesz (és lefagy...☹)

Fontos, hogy ez az emuláció korlátozottabb körű emuláció, mint amiről idáig szó volt, itt lényegében csak a privilegizált utasításokat interpretálja a VMM, ezeket is a CPU hardveresen találja meg, nem kell a futtatható bináris kódot szoftveresen feldolgozni

Szoftveres virtualizáció – példa



Forrás: Carl Waldspurger, Introduction to Virtual Machines



24



Példa forrása: <http://labs.vmware.com/academic/mit-iap-2010>

A vendégben futó kód:

- Az első utasítás egyik regisztert mozgat a másikba, ezzel nincs gond
- cli (clear interrupt flag): nem engedhetjük meg, hogy a fizikai gépen tiltsa le a megszakításokat, így ehelyett egy más utasítást hajtunk végre. Itt csak in-line módon lecseréljük egy másik utasításra.
- A cr3 regiszter a laptábla kezdőcímét jelzi, így a virtuális gépben futó kód ezt sem módosíthatja. Itt az utasítás lecserélésén túl a VMM egy külön rutinját is meghívjuk, ami ezt lekezeli (HANDLE_CR3).

Szoftveres virtualizáció

- Mit tegyünk a problémás utasításokkal?
- Szoftveres virtualizáció bináris fordítással
 - (a.k.a Binary Translation - VMware)
 - Egy JIT fordító a végrehajtás előtt végignézi a kód szegmenst és kicseréli a problémás utasításokat pl. SYSCALL-ra, vagy valami egyéb kódrészletre
 - Kicserélhet egyéb, amúgy elfogható utasításokat is rögtön a kezelő kódrészlettel, hogy elkerülje a felesleges hívást a VMM-be (inline translation)
 - Mivel a kód hossza megváltozhat ezért a kódra mutató pointereket (jump, branch utasítások) is módosítani kell
 - Gyors: nem teljes fordítást végez, az utasítások többségét változatlanul hagyja
 - Gyors2: a JIT fordító minden kódrészletet csak az első futáskor jár be, ismételt futtatáskor már cache-eli a már módosított kódrészleteket
 - Gyors3: optimalizálással csökkenthető a VMM-be történő hívások száma
 - A végrehajtás menete: elő-fordítás + virtualizált végrehajtás elfogással + elfogott utasítások emulációja



25



Fontos, hogy a teljes CPU emulációnál használt futtatókörnyezettől eltérően itt csak a kódra mutató pointereket kell átírni (futási időben csak önmódosító kód esetén változhat), az adatra mutató pointerek (amik futási időben változnak!) változatlanul hagyhatók. Ezért a virtuális gép futása elő-fordítási és futási fázisokra bomlik. Az előfordítási lépések mindig akkor történnek, ha új (korábban még nem futtatott) kódrészlethez kerülne a vezérlés. Az előfordító vezérlése „lusta” módon történik, csak akkor hívódik meg, ha feltétlen szükség van rá és nem dolgozik nagyon előre ezért az előfordítási fázisok külön-külön rövid ideig tartanak -> nem rontja le nagyon a virtuális gép válaszidejét.

Bináris fordítás működése példával illusztrálva:

Keith Adams, Ole Agesen: A comparison of software and hardware techniques for x86 virtualization

<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.117.196&rep=rep1&type=pdf>

Három lehetőség a virtualizációra

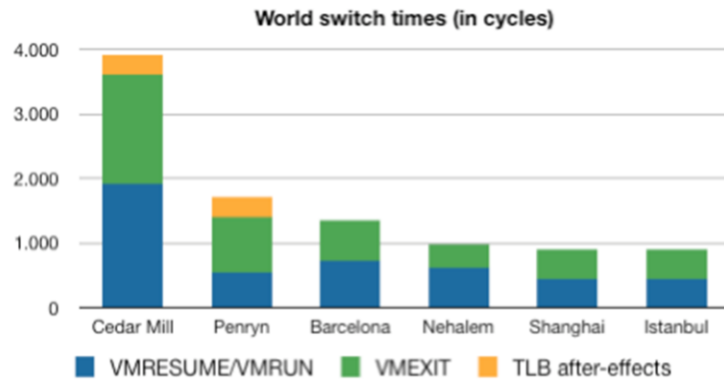
- **Virtualizáció** – az utasításokat (egy részüket) változatlanul hagyja végrehajtani, csak a problémás privilegizáltakkal kell valamit kezdeni
 - **Szoftveres virtualizáció** (Trap and Emulate + bináris fordítás)
 - **Hardveres virtualizáció** (Trap and Emulate, teljesen hardveres támogatással)
 - **Paravirtualizáció** (vendég OS kódját módosítjuk, hogy tudjon a virtualizációról, és hívhassa a VMM-et)

Hardveres virtualizáció



- Intel VT-x és AMD-V kiegészítésekkel új mód jött be (hogyan uralkodjon mind felett...)
- SYSCALL és INT Ring3-ból Ring0-ba hív át – nem kell felesleges kört futnia a VMM-ben
- Van külön VMCALL utasítás is, amivel ki lehet hívni a Root Mode-ba
- Minden szükséges privilegizált utasítást elkap
- De ennek nem feltétlen örülünk... a binary translation sok optimalizációra adott lehetőséget, ami itt hiányzik
- Lassabb volt a szoftveres virtualizációnál (de javul, a VMCALL-VMRESUME körülfordulási időt minden CPU generációval csökkentik)

Hardveres virtualizáció



28



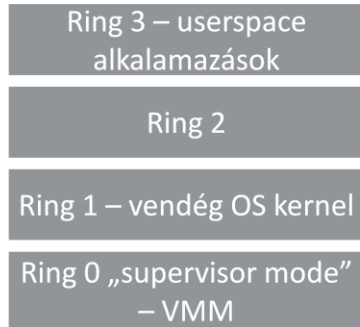
Képek forrása:

1. Intel
2. Anandtech – Optimizing for virtualization <http://www.anandtech.com/show/2786>

Három lehetőség a virtualizációra

- **Virtualizáció** – az utasításokat (egy részüket) változatlanul hagyja végrehajtani, csak a problémás privilegizáltakkal kell valamit kezdeni
 - **Szoftveres virtualizáció** (Trap and Emulate + bináris fordítás)
 - **Hardveres virtualizáció** (Trap and Emulate, teljesen hardveres támogatással)
 - **Paravirtualizáció** (vendég OS kódját módosítjuk, hogy tudjon a virtualizációról, és hívhassa a VMM-et)

Paravirtualizáció



- Minek ez a sok hajcíhő?!
- Módosítsuk a vendég OS kernelt, hogy ne használjon elfoghatatlan utasításokat! (Feltéve, hogy megtehetjük...)
- Nem kell semmiféle előfordító
- A vendég OS tehát kifejezetten tud róla, hogy virtualizált
- Miért állnánk meg itt?
- Ne csak az elfoghatatlan, de minden „szükségtelen” vagy „elkerülhető” privilegizált utasítást irtsunk ki a vendég kernelből. – Kevesebb váltás kell a VMM-be.
- Vezessünk be saját rendszerhívásokat a vendég kernel-VMM kommunikációra, és amit csak lehet, ezzel oldjuk meg (perifériák kezelése)



Mivel a vendég gépet módosítjuk, hogy együttműködjön egy virtualizációs környezettel ezért sokkal egyszerűbbé válik a VMM megvalósítása is. DE ez nem jelenti azt, hogy a VMM védelmét csökkenthetnénk, nem tételezhetjük fel, hogy a vendég OS mindig be fogja tartani a szabályokat, meghibásodhat, vagy támadó céllal módosíthatják a működését. A különbség annyi, hogy paravirtualizációnál a VMM nem kell, hogy megpróbálja emulálni a hatását olyan műveleteknek, amiket a vendég OS megegyezés szerint nem adhatna ki, hanem – el nem fogható műveletnél - hagyja összeomlani, vagy – elfogható, de nem engedélyezett műveletnél - explicit leállítja a vendég gépet.

Tartalom

Platform virtualizáció technikai háttere

- Ismétlés: mit csinál egy CPU
- Mi kell egy virtuális CPU-hoz (Popek & Goldberg)
- Szimuláció, emuláció, virtualizáció
- Három lehetőség a virtualizációra: szoftveres, hardveres, para-
- **Memóriakezelés egy modern CPU-ban – innen folytatjuk a következő előadáson**
- Virtuális gépek memóriakezelése: szoftveres, hardveres, para-
- Virtuális memóriakezelés speciális képességei: megosztás, késleltetett allokáció, memória-ballon

További információ

- O. Agesen, A. Garthwaite, J. Sheldon, and P. Subrahmanyam, "The evolution of an x86 virtual machine monitor," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 4, p. 3, Dec. 2010.
<http://dl.acm.org/citation.cfm?id=1899930>
- P. Barham et al., "Xen and the art of virtualization," *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 164-177, 2003.
<http://dl.acm.org/citation.cfm?id=1165389.945462>
- K. Adams and O. Agesen, "A comparison of software and hardware techniques for x86 virtualization," in *Proc. of the 12th Int. Conf. on Architectural support for programming languages and operating systems*, San Jose, California, USA, 2006, pp. 2-13.
<http://dl.acm.org/citation.cfm?id=1168860>



A cikkek BME-s hálózaton belülről elérhetők és letölthetők a BME könyvtári előfizetéseinek köszönhetően (lásd: <http://www.omikk.bme.hu/>)!