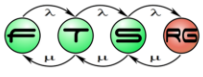
Virtualizációs Technológiák és Alkalmazásaik

CPU és memória virtualizáció

Tóth Dániel

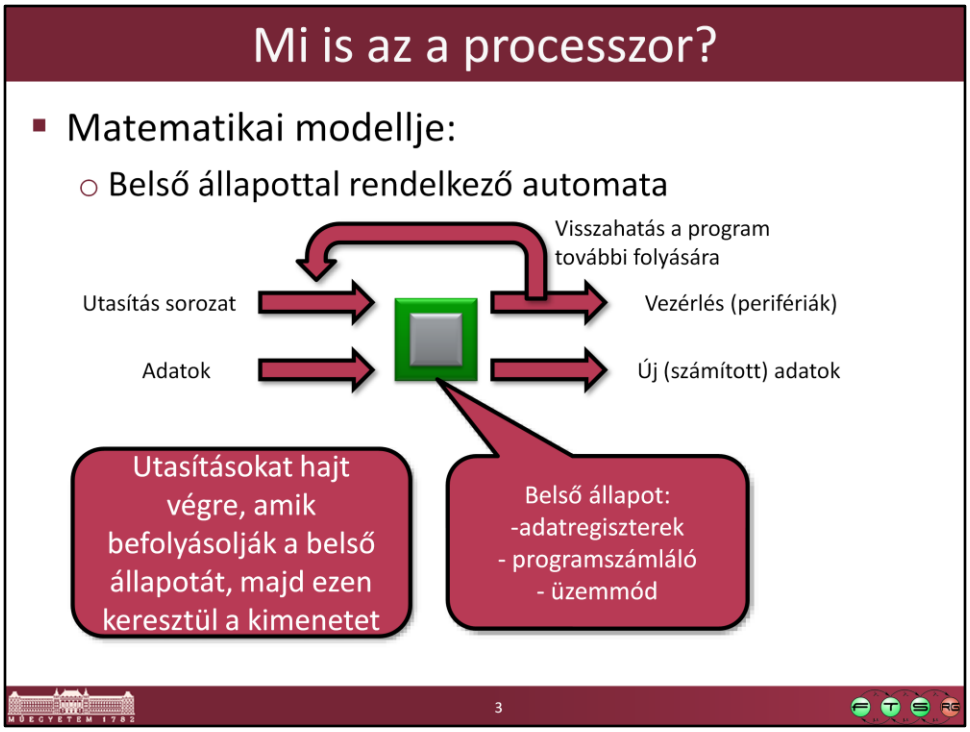


Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

Utolsó módosítás:2010. 09. 15.

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



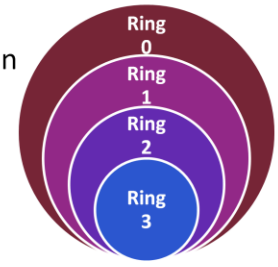
Kicsit konkrétabban: az utasítás hatására a belső regiszterek valamelyikének értékét módosítja, felhasználva regiszter értékeket és/vagy kívülről betöltött adatot. A belső regiszter tartalma lehet később kimenő adat, vagy befolyásolhatja a program további futását. Pl. a programszámlálót átírja, ezzel 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 nyelven használható 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 tört, logikai stb.)
    - speciális regiszterek, pl. programszámláló, verem mutató (stack pointer)
  - CPU üzemmódjait
    - Minden üzemmódban más-más részhalmaza engedélyezett az utasításkészletnek
    - Memória elérése lényegesen eltérhet üzemmódok között

Mit tud processzor?

- Az ISA specifikáció biztosítja, hogy egy program - azonos bemenő adattal ellátva - minden esetben ugyanúgy fusson le, ugyanazt az eredmény adja
- A CPU üzemmódok célja:
  - Visszamenőleges kompatibilitás (x86 másból sem áll…)
  - Egy program ne tudjon bizonyos műveleteket végezni
  - Operációs rendszer el tudja szigetelni a programokat egymástól
    - „védett” módok
    - Futási privilégium szintek, al-üzemmódok (rings)
- Példa: 4 privilégiumszint az x86 védett módjában
  - Ring 0, *supervisor* mód: legbővebb utasításhalmaz
  - Ring 1
  - Ring 2
  - Ring 3, *user* mód: legszűkebb részhalmaz



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 tartalmaznak egy futási szintet (DPL – descriptor privilege level). Hozzáférés (olvasás, írás, végrehajtás) általános esetben akkor engedélyezett, ha a CPL <= 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 Manual4-9 fejezet.
http://download.intel.com/design/PentiumII/manuals/24319202.pdf

Mit tud processzor?

- Eseménykezelés
  - Megszakítás (interrupt) beérkezése azonnali állapotváltással jár
    - A megszakítás száma (csatornája) alapján a CPU egy meghatározott címen folytatja a végrehajtást
    - Lehet hardveres (perifériától származó) és szoftveres eredetű (INT utasítás x86-on)
    - Interrupt vector – egy tömb ami tartalmazza a megszakítás számához tartozó lekezelő programrészletre mutató pointert
    - Megszakítás válthatja a CPU üzemmódját is (user módból máshogy nem is lehet kikerülni)
    - A szoftveres megszakításból alakultak ki a modernebb rendszerhívás (syscall) utasítások

Az x86 esetén vannak egyéb CPU üzemmódváltási lehetőségek is, pl Call Gate leírón keresztüli kód szegmensbe hívásnál RPL (Requested Protection Level) mező, de modern operációs rendszerek nem támaszkodnak erre a megoldásra.

Mit tud processzor?

- Eseménykezelés II.
  - Ha a programban olyan utasítás kerül sorra, ami az adott üzemmódban nem engedélyezett, akkor kivétel keletkezik (Illegal Instruction *Exception*, vagy *Trap*)
  - A kivétel egy speciális fajta szoftveres megszakítás
  - Ilyenkor is váltás történik a supervisor módba és a vezérlés egy megadott memóriaterületen lévő lekezelő rutinhoz kerül
- Az operációs rendszerek alapvető része a megszakításkezelési keretrendszer
  - Az alkalmazások szoftveres megszakítással vagy rendszerhívással kérhetik meg a rendszermagot különböző műveletekre
  - A rendszermag feladata a perifériáktól érkező események fogadása is
  - Az időzítő periféria periodikus megszakításküldésével jut vissza a vezérlés rendszeresen a rendszermaghoz, így működnek OS ütemezők
  - A rendszermag mindig értesül a user módban futó folyamatok „tilosban járásairól” is

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, mint ha fizikai CPU-n közvetlenül futna
- **Erőforrás kezelés**
  - A VMM minden virtualizált erőforrást teljes egészében maga felügyel
- **Hatékonyság**
  - A virtuális gépben futó program utasításainak nagy része változtatás és VMM beavatkozás nélkül fut a fizikai CPU-n

Mi teljesül ezekből egy OS felett futó folyamatnál?

És mi a helyzet, ha OS fut VMM felett?

* Gerald J. Popek and Robert P. Goldberg: Formal Requirements for Virtualizable Third Generation Architectures, 1974

Virtuális CPU feltételei

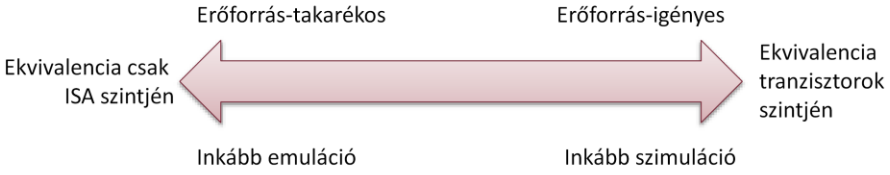
- Mik a problémák
  - Az OS felett futó folyamat korlátozottabb CPU üzemmódban fut, közvetlenül nem menedzsel hardver erőforrásokat, nem futhat OS nélkül – ez nem ekvivalens futási környezet
  - OS nem futhat user módban, mert hardver erőforrásokat kell közvetlenül menedzselnie, ehhez kellenek a supervisor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

Tartalom

- Platform virtualizáció technikai háttere
  - 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

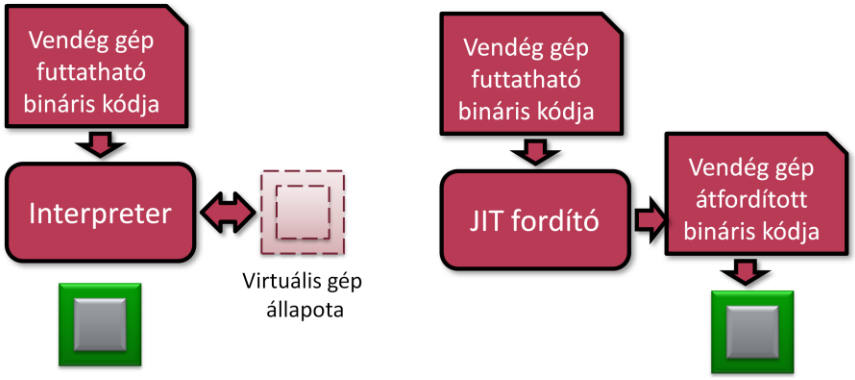
- Ekvivalencia biztosítása
  - **Szimuláció**: szoftverrel modellezzük a processzor belső működését (különböző mélységig megtartva a valósághűséget). Fizikailag sohasem fut a virtuális gép kódja a CPU-n.
  - **Emuláció**: helyettesítjük a VMM felett futtatandó programot/annak egy részletét egy másikkal, 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-étól (sőt azonos architektúra emulációnál is eltérő utasítás részhalmazt használ!)
  - Virtualizációhoz (P&G értelemben vett) képest lassú



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 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)*

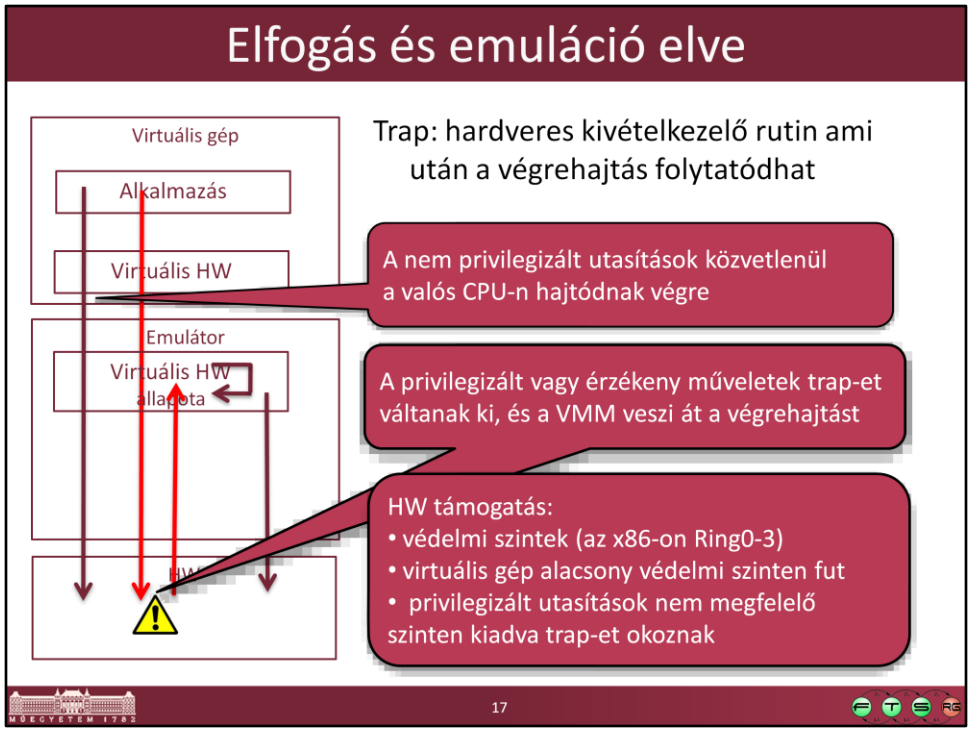
Interpreter hogyan hordozható: ha pl. C nyelven írjuk (bizonyos kódolási konvenciók betartásával), akkor bármely célplatformon futtatható, amire van C fordító
JIT fordító viszont specifikus kell hogy legyen minden virtuális gép-futtató platform párra.
A Java VM és a .NET CLR is Jit fordítós emulátorok, olyan architektúrát emulálnak, ami fizikai processzoron sohasem létezett. Viszont ezek alkalmazásfuttató környezetek, NEM tartoznak a platform virtualizáció témakörébe!

Tartalom

- Platform virtualizáció technikai háttere
  - 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

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**

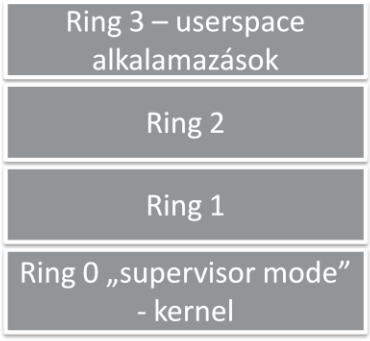


Virtuális gép alacsonyabb védelmi szinten futtatása: deprivilegizálás (de-privileging)
x86 ring: 4 darab van, 0 a legmagasabb (itt futnak általában az OS-ek kerneljei), 3 a legalacsonyabb (itt futnak az alkalmazások)
A processzor trap művelete kb olyan, mint egy kivétel (exception): van egy „catch” blokkja, 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.)

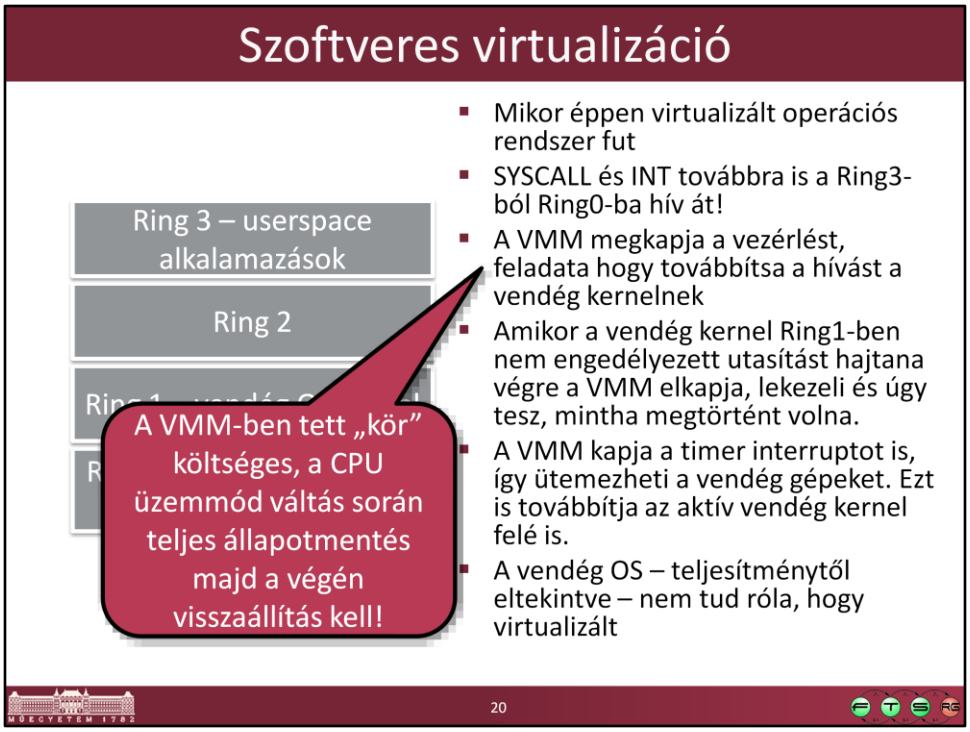
Elfogás és emuláció

- 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

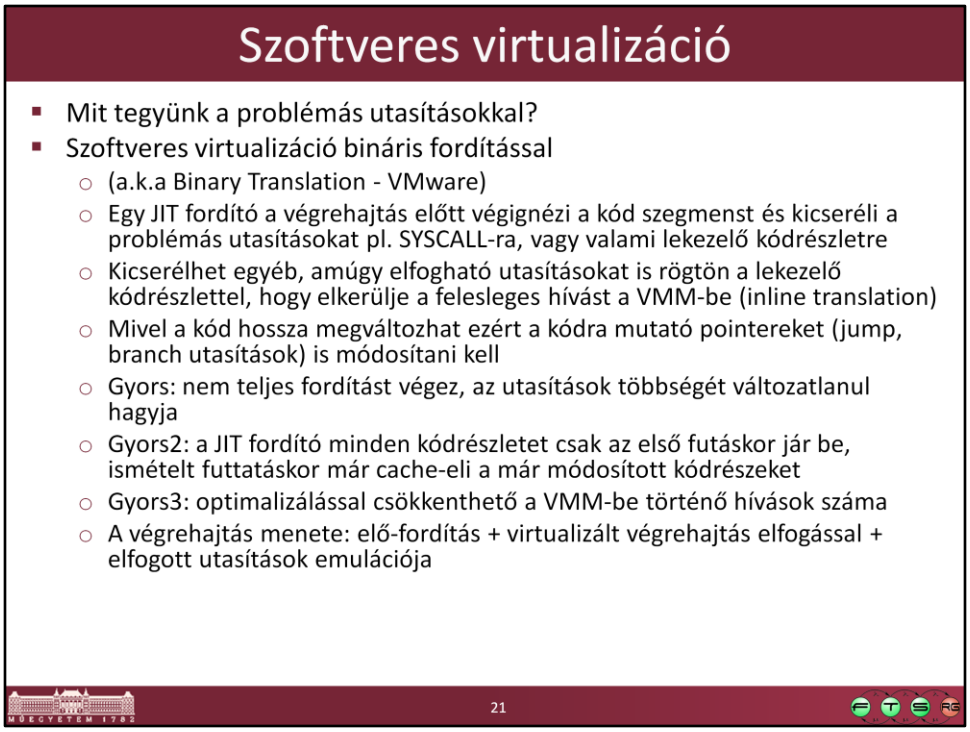
Oprendszerek működése



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



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



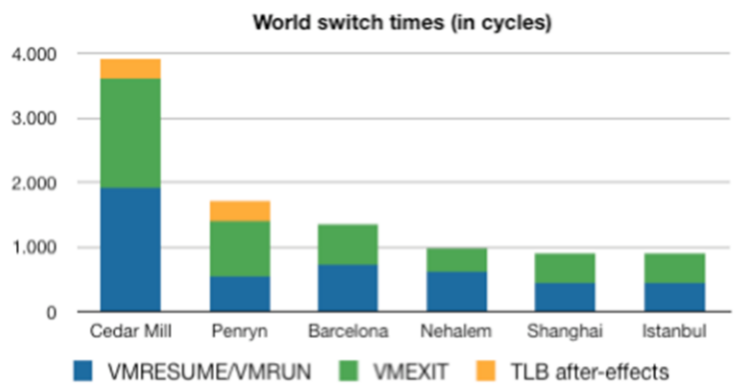
Fontos, hogy a teljes CPU emulációnál használt futtatókörnyezettől eltérően itt csak a kódra mutató pointereket kell átírni (futási időben csak önmódosító kód esetén változhat), az adatra mutató pointerek (amik futási időben változnak!) változatlanul hagyhatók. Ezért a virtuális gép futása elő-fordítási 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

Hardveres virtualizáció

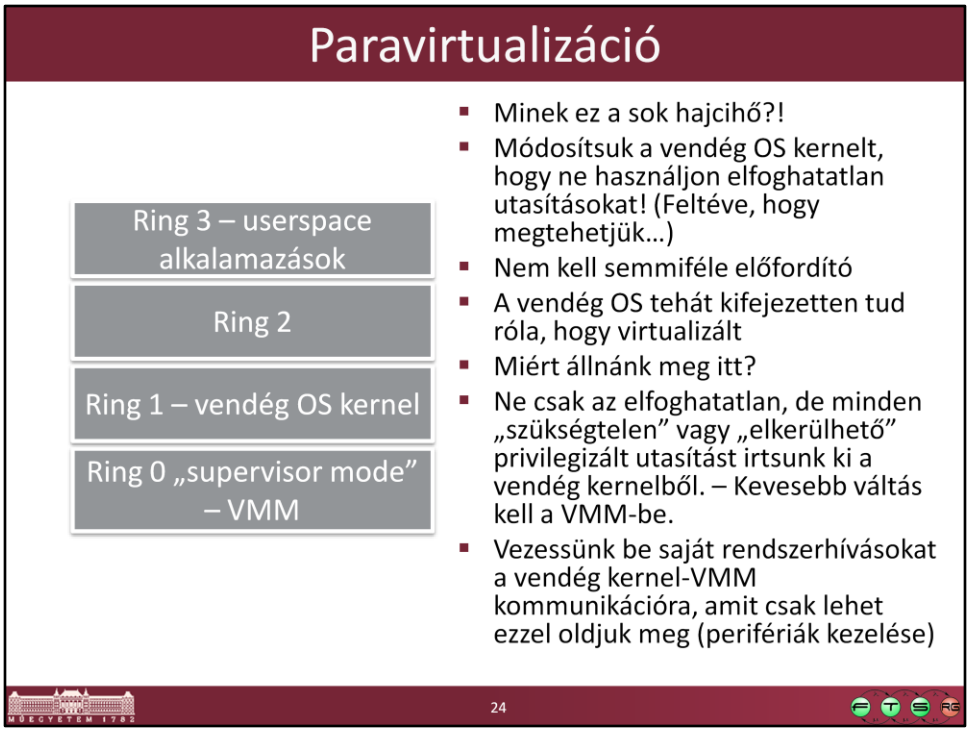
- Intel VT-x és AMD-V kiegészítésekkel új ring jött be (hogy uralkodjon mind felett…)
- SYSCALL és INT Ring3-ból Ring0-ba hív át – nem kell felesleges kört futnia a VMM-ben
- Van külön VMCALL utasítás is, amivel ki lehet hívni a Root Mode-ba
- Minden szükséges privilegizált utasítást elkap
- De ennek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

Hardveres virtualizáció



Képek forrása:

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



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 célzattal 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