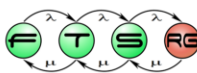


Memória és perifériák virtualizációja

Tóth Dániel



Utolsó módosítás: 2010. 09. 22.

Tartalom

- Előző rész tartalmából:
 - CPU virtualizáció
 - A három alap virtualizációs megközelítés
- Memória virtualizáció
 - Virtuális memória az operációs rendszerekben
 - Virtuális memória a platform virtualizációban
 - Virtuális memóriakezelés speciális képességei: megosztás, késleltetett allokáció, memória-ballon
- Perifériák virtualizációja
 - Perifériák programozói felülete általában
 - Periféria virtualizációs architektúrák

Emlékeztető: 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
- **Hatékonyág**
 - A virtuális gépben futó program utasításainak nagy része változtatás és VMM beavatkozás nélkül fut a fizikai CPU-n



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

Emlékeztető: A három virtualizációs lehetőség

- **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**

Tartalom

- Előző rész tartalmából:
 - CPU virtualizáció
 - A három alap virtualizációs megközelítés
- Memória virtualizáció
 - **Virtuális memória az operációs rendszerekben**
 - Virtuális memória a platform virtualizációban
 - Virtuális memóriakezelés speciális képességei: megosztás, késleltetett allokáció, memória-ballon
- Perifériák virtualizációja
 - Perifériák programozói felülete általában
 - Periféria virtualizációs architektúrák

Virtuális memóriakezelés

- Modern CPU-k tartalmaznak memóriakezelő egységet (MMU – *memory management unit*)
 - Feladata „virtuális” memóriacímeket leképezni „fizikaira”
 - Mi is az a virtuális memóriacím? Hol használható?
 - CPU felhasználói (pl. Ring 1-3) módjaiban virtuális címekkel dolgozik (nem feltétlenül, de a modern OS-eknél ez igaz)
 - Az alkalmazások nem a fizikai memóriacímeket látják
 - Cél: áthelyezhető legyen az operációs rendszer felett futó alkalmazások kódja, ne csak fix beárazott helyen tudjon futni (akár szoftveresen is megoldható lenne...)
 - Cél2: eközben a teljesítmény ne romoljon számottevően (ehhez már hardver támogatás is kell)



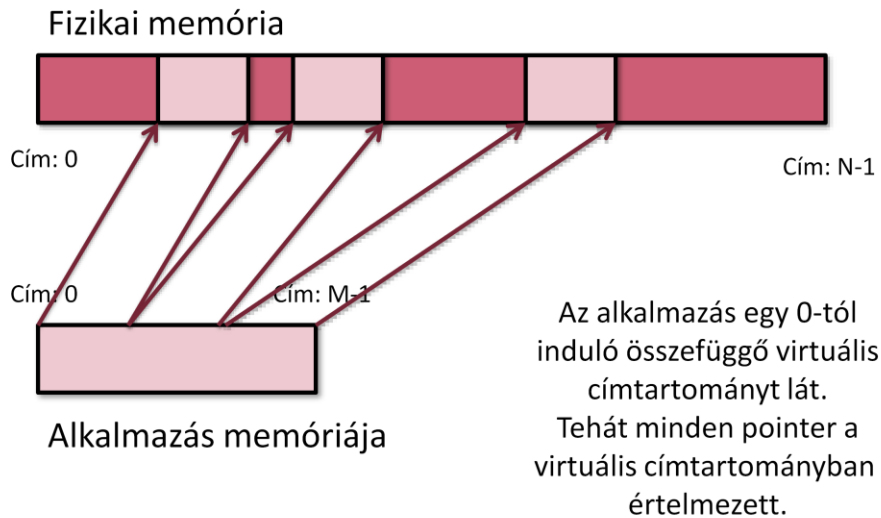
MMU != fizikai memóriaillesztő áramkör. Az utóbbinak a feladata a memória modulok elektromos alacsony szintű vezérlése, ez sokáig a CPU-n kívül a chipset „északi hídban” foglalt helyet, csak a közelmúltban (Athlon64, Core i7) került be a processzorba. A memóriaillesztő áramkör is végez címfordítást, de ez már teljesen láthatatlan a CPU és a perifériák számára. A fizikai memóriacím alatt a továbbiakban azt értjük amit a CPU és/vagy perifériák kiadnak a buszokon.

x86-ra jellemző, hogy kétféle virtuális memóriacím kezelési mechanizmus van: szegmens alapú és lapozott. A szegmens alapú 32 bites üzemmódban _mindig_ be van kapcsolva, de egy szegmens tartalmazhatja a teljes memóriát. 64 bites üzemmódban viszont szegmens alapú memóriaszervezés nincs. A lapozott üzemmód az aktuális ringtől függetlenül lehet ki vagy bekapcsolt állapotban, de átkapcsolni csak ring 0-ban lehet. Manapság tipikusan csak a kernel használ fizikai memóriacímeket (nem feltétlenül az összes funkciójához), az alkalmazások mindig lapozott memóriát használnak.

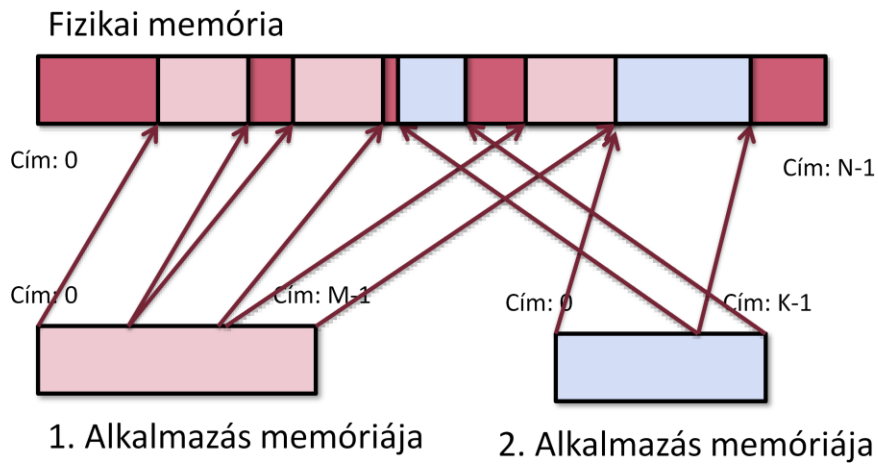
Részletek:

Intel® 64 and IA-32 Architectures Software Developer's Manual
<http://www.intel.com/products/processor/manuals/>

Virtuális memóriakezelés



Virtuális memóriakezelés

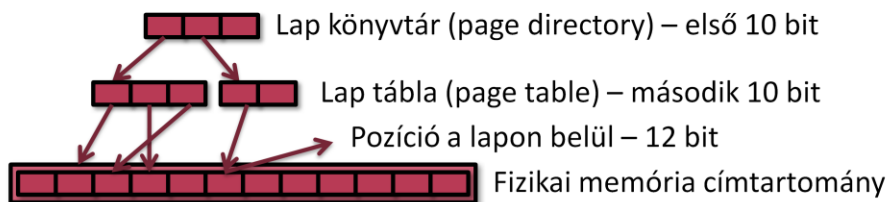


Ez minden alkalmazásra igaz

Virtuális memória megvalósítása lapokkal

■ Memória lapok (*pages*)

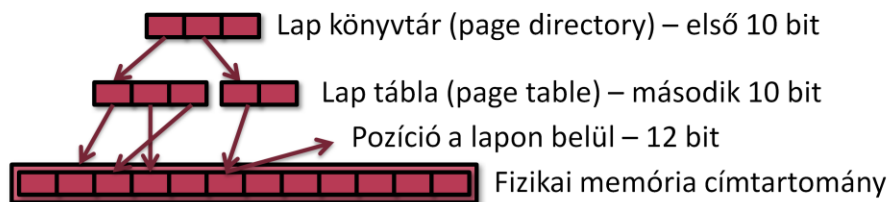
- Virtuális → fizikai memória cím hozzárendelés
- Tipikusan (x86) 4 kB méretű allokációs egységekben
- A cím utolsó 12 bitje a lapon belüli cím
- A cím első 20 bitje kétszintű (10-10 bit) laptábla cím
 - Létezik óriás lap üzemmód is, ilyenkor csak egyszintű laptábla van, ezen belül 22 bit (4MB) pozíciócím



Megjegyzés: a felsorolt konkrét számadatok x86 architektúrán érvényesek, más architektúrán előfordul más beosztás is. Az x86-on is léteznek még speciális üzemmódok (PAE – paging area extension, NX – no execute), amik 1-1 felső helyiértékű bitet saját célra használnak.

Virtuális memória megvalósítása lapokkal

- A laptáblák is a fizikai memóriában foglalnak helyet
 - Csak az operációs rendszer kernel módosíthatja őket
 - Minden alkalmazáshoz másik táblakészlet tartozik, a kernel ütemezéskor cseréli ki mindig a megfelelőre
 - Az MMU a CPU laptábla regisztere alapján tudja, hogy hol kell keresni legfelső szintű lap könyvtárat
 - Automatikusan feloldja a virtuális címeket fizikaira, a virtuális címeket használó kód módosítás nélkül fut
 - A virtuális címtartományból kicímzés vagy read-only bittel jelölt lapra írás trap-et vált ki a CPU-ban



A trap mechanizmus teszi lehetővé, hogy az éppen nem használt lapokat diszkre lehessen menteni majd vissza behúzni a memóriába akkor, amikor éppen hozzáférés történik. A hozzáférés a nem létező lapra trap-et vált ki, ezt elfogja és ekkor teszi vissza a kernel a lapot a swap területről.

Memória virtualizálása

- A Ring 0-tól eltérő szinteken futó folyamatok virtuális memóriát látnak
 - A virtuális -> fizikai cím feloldása hardverben történik laptáblák alapján. Gyors, TLB cache-eli fizikai-virtuális cím hozzárendelést.
- Használhatjuk ezt a vendég gépek memóriájához?
 - Több szint kell: a VM-ben is kell saját laptábla a saját alkalmazásokhoz
 - De a CPU ilyen nem támogat.
 - Lapszervezés kombinálható a szegmens szervezéssel („gányolás” de x86 szoftveres virtualizációnál néhány helyen jól jön)
 - Kell két példány a laptáblából az egyik a vendég kernel számára látható és a vendég VM „fizikai” címeit képezi le a VM-en belül futó alkalmazások „virtuális” címére.
A másik a VMM (és a CPU) számára látható ún. *árnyék laptábla* (*Shadow page table*) és a futtató gép „igazi fizikai” címeit képezi le a VM-en belül futó alkalmazások virtuális címére.



11



Innentől a fizikai memória fogalmánál meg kell különböztetni a hoszt gép szempontjából fizikai memóriacímeket és guest operációs rendszer által „fizikai” memóriának látott címeket, amik valójában a hoszt gép szempontjából már virtuálisak. Tehát az idézőjelnek itt most jelentésmegkülönböztető szerepe van. :)

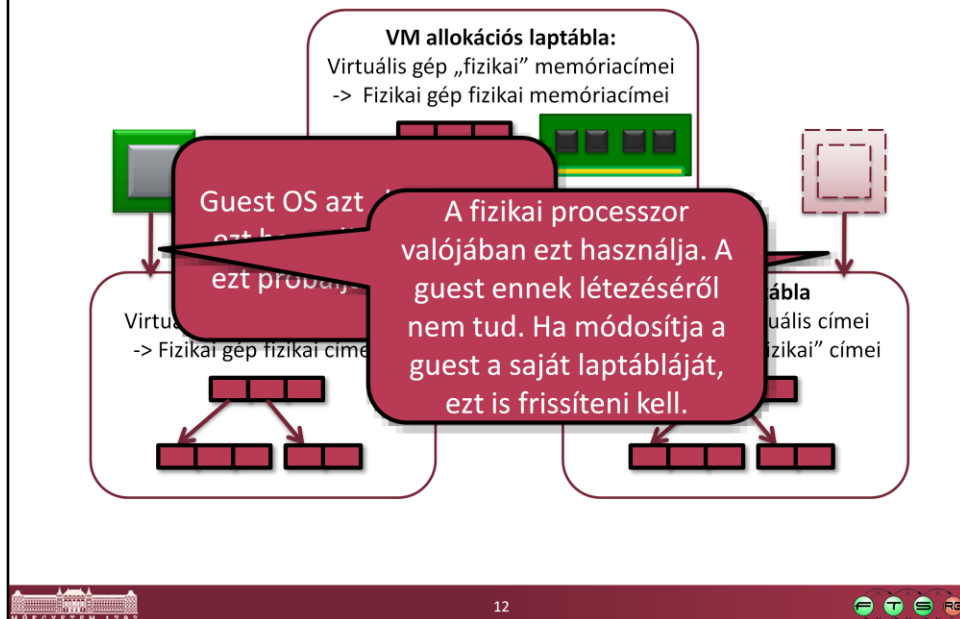
Érdekesség, hogy x86-on a szegmens szervezés teszi lehetővé, hogy a guest-tel azonos memóriaterományban, de mégis a guest beavatkozásától védett módon legyenek elhelyezhetőek a VMM kezelésében lévő memóriaterületek. Például a JIT fordított kódterületek ilyen elhelyezésével lehet gyors (trap és kontextusváltás nélküli) áthívás a kétféle memóriaterület között. Viszont 64 bites üzemmódban a szegmens alapú védelem nem működik, ez a legfőbb oka annak, hogy x86-64 bites üzemmódot bináris fordítással a legtöbb virtualizációs szoftverben nem építettek ki.

Az AMD egyes processzoraiban éppen emiatt viszont korlátozottan mégis van szegmens alapú védelem 64 bites üzemmódban is, de ez nem túl széles körben támogatott.

Részletek:

Keith Adams, Ole Agesen: A Comparison of Software and Hardware Techniques for x86. Virtualization.

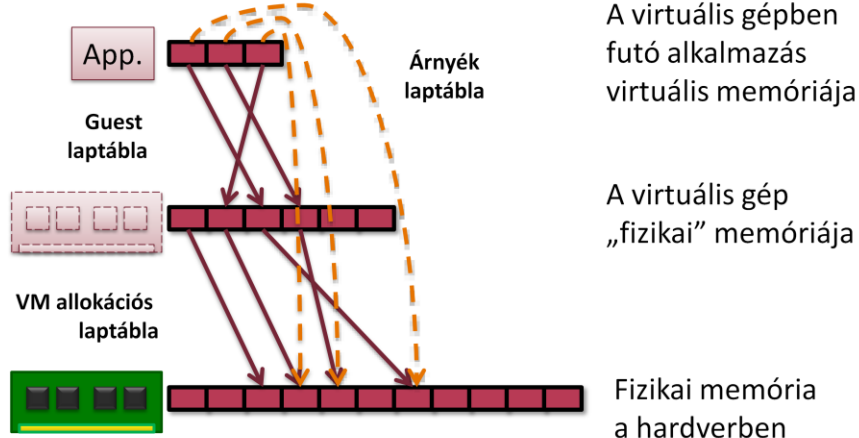
Memória virtualizálása



Az árnyék láptábla szerepe, hogy az eredetileg kétszintű indirekcióból egyszintű indirekciót csináljon.

Memória virtualizálása

▪ Másik nézetben:



Memória virtualizálása

- De mi van, ha guest kernel módosítani akarja a laptábláját?
 - Megfelelően frissíteni kell az árnyék táblát is
 - -> természetesen Trap and emulate, de hogyan?

Tanulságok:

- A memóriakezelésben is a háromféle fő megvalósítás megtalálható.
- A memóriakezeléshez extra laptáblák kellenek, tehát a VM több host memóriát igényel, mint amennyi számára látható lesz
- A memória foglalása és felszabadítása extra feladatot jelenthet a VMM számára is (kivéve hardveresen virtualizált MMU-nál)
- Az egész árnyék tábla frissítési problémát hardveresen kezel
- Harmadik megoldás: „természetesen?! trap and emulate?”
 - A guest kernel egyszerűen ne maga akarja módosítani a laptáblát, *kérje meg* a VMM-et erre 😊



14



A hardveres laptábla kezelés elég sok előnnyel jár, ezzel már a hardveres virtualizáció egyértelműen gyorsabb lehet a bináris fordításnál. Elkerülhetőek vele a kontextusváltásnál a TLB ürítések, a VMM-ben tett körök a memória allokáció változtatásánál stb.

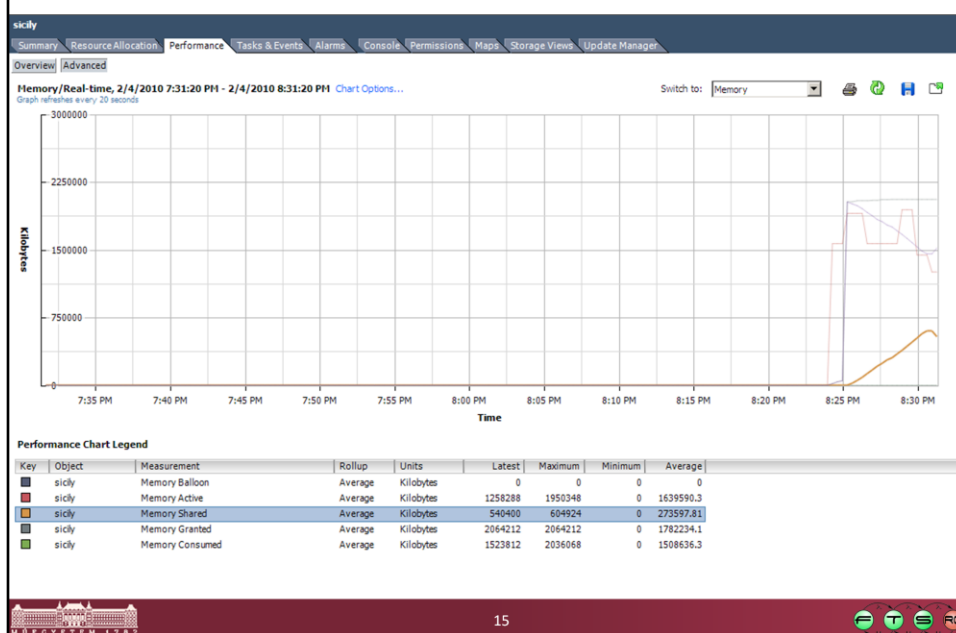
Intel megvalósításának részletes leírása:

<http://www.intel.com/products/processor/manuals/>

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3B: System Programming Guide, chapter 25.2

Aki még nem jött volna rá, a 3. megoldást természetesen paravirtualizációs megközelítésnek nevezik.

Extra memória virtualizálási lehetőségek



A nem túl jól látható (éles gépen éppen elcsúszott) képernyőképen az látható, hogy az indítás után a guest gép memóriáját fokozatosan pásztázza végig a VMM, megosztható memórialapokat keresve, így fokozatosan emelkedik a shared memória mennyisége.

Extra memória virtualizálási lehetőségek

- Dinamikus allokáció
 - ami memóriát nem használ ki a vendég, arra ne is tartsunk fenn lapokat
 - Gyakorlati haszna önmagában elenyésző, a legtöbb OS az összes szabad memóriát disk cache-nek használja

Tanulságok:

- - Figyelni kell a memória foglaltságot, nagyot esik a teljesítmény, ha diszkre kell lapozni
 - Ha van rá lehetőség, ki kell használni a vendégbe telepíthető ballooning drivert, elkerülhető vele a vergődés
- Egyrészt az agens által „fogott” memórialapok mögé nem is allokál fizikai memóriát, így nyer vissza helyet
- Egyrészt a vendég fel fog adni a disk cache-ből,
- Másrészt el fog kezdeni kilapozni a saját swap területre, elkerüli, hogy a host is swappeljen



16



Miért olyan rossz az, ha a host swappel? Mert a guest nem tudja, hogy a memóriatarományának egy része diszken van tehát lassú, ezt a területet disk cache-nek vagy alkalmazásoknak fogja használni, amivel még többet árt. Esetleg ha a host és a guest is swappel egyszerre, akkor lehet, hogy a guest a saját swap területéről a (memóriának képzelt) host swap területére fog mozgatni adatot és viszont -> mega trashelés. Ha csak a guest swappel, akkor ez a helyzet nem fordulhat elő.

Tartalom

- Előző rész tartalmából:
 - CPU virtualizáció
 - A három alap virtualizációs megközelítés
- Memória virtualizáció
 - Virtuális memória az operációs rendszerekben
 - Virtuális memória a platform virtualizációban
 - Virtuális memóriakezelés speciális képességei: megosztás, késleltetett allokáció, memória-ballon
- Perifériák virtualizációja
 - Perifériák programozói felülete általában
 - Periféria virtualizációs architektúrák

Perifériákról általában

- A perifériák kezelése jellegzetesen
 - CPU felprogramozza a perifériát, regiszterek átírása
 - Periféria eseményt jelez a CPU felé, megszakítás
 - Ilyenkor valamilyen módon le kell kezelni az eseményt, valamit reagálni kell rá
 - Periféria maga elvégzi a feladatát, közvetlen memória hozzáférés
 - Kiolvas elküldendő adatot, vagy berak beérkezőt
 - Külön lefoglalt fizikai memóriaterület kell e

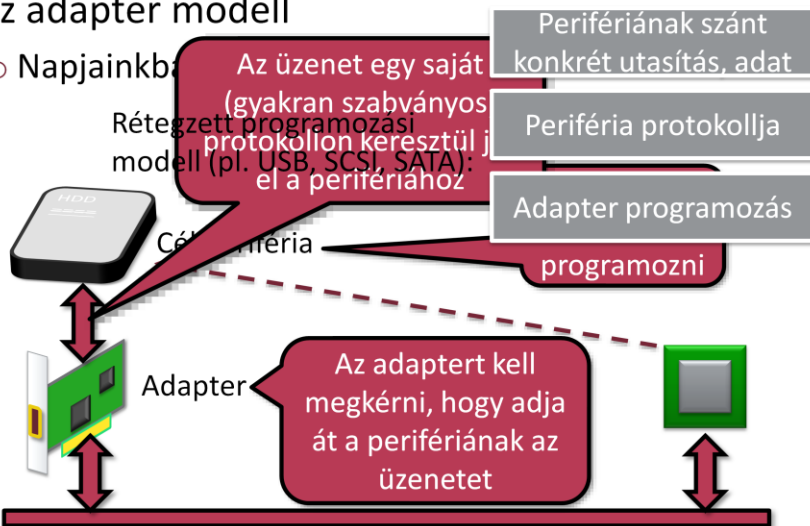
A periféria kezelése a „driver” felelőssége, ami többé kevésbé része az OS kernelnek



Perifériákról általában

■ Az adapter modell

○ Napjainkban



Perifériák virtualizációja

■ Emuláció

- Trap and emulate -> az I/O műveleteket kell elfogni
 - Ez adódik a CPU futási szintjéből, Ring1-3-ban az I/O műveleteket elfogja
 - Ha a periféria memóriatartományba illesztett, akkor read-only memórialappal fogható el a felprogramozása
- Valamilyen létező fajta hardver működését (regiszterek, megszakítás, DMA) emulálni kell egy szoftveres komponenssel
 - Egyfajta „anti-driver”, általános elnevezése *Backend*
 - A vendégben futó meghajtóprogram valójában ezzel beszélget.
- Minden I/O művelet egy kör VMM-ben -> lassú

Perifériák virtualizációja

■ Paravirtualizáció

- Egyszerűsítsük az emulált hardvert, tervezzünk „nem létező fajta” hardvert, amit a legkevesebb művelettel lehet vezérelni
- Egy összetett művelet akár csak egy VMM hívást igényel

Tanulságok:

- I/O intenzív alkalmazásoknál számolni kell jelentős teljesítményvesztéssel
- Ha van rá lehetőség telepítsük fel a paravirtualizált eszközmeghajtókat a vendég operációs rendszerbe
- Lehet olyan feladat, amit közvetlenül a virtuális géphez rendelt hardverrel célszerű megoldani (backup szerver, 3D gyorsítás...)

elérhető

- Léteznek IOMMU megoldások is (pl.: Intel VT-d), használata terjed
- Léteznek olyan hardverek, amik képesek több guest felől is függetlenül, belső állapot konzisztens megtartással kéréseket fogadni (SR-IOV)



21



Fontos tanulság megint: amikor a paravirtualizációnál magasszintű erőforrásokat adunk át a guest OS-nek, akkor látható, hogy itt is - mint mindenhol máshol is – folytonos átmenet kezd kialakulni az operációs rendszerek és virtualizációs környezetek között. Tehát a szigorú taxonomizálás valójában megint csak egy „modell”, ami a megértést segíti, de a valóságot kicsit absztrahálja.

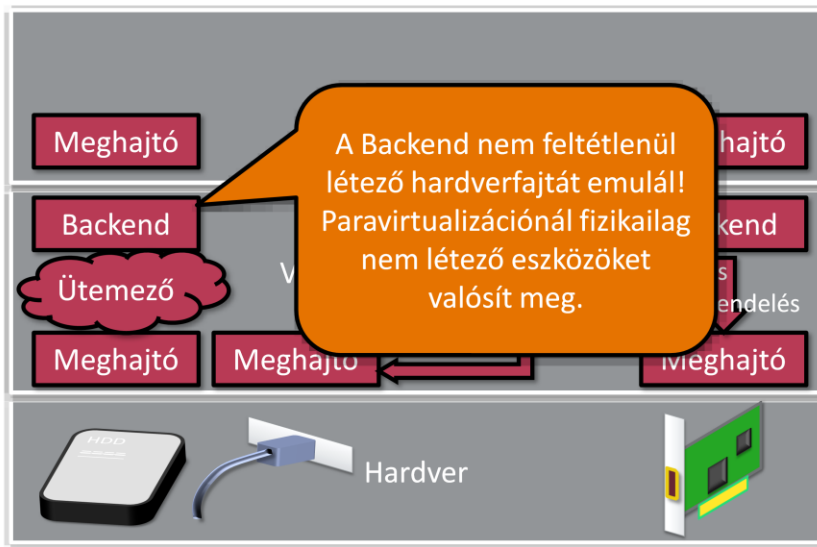
SR-IOV, manapság főleg hálózati adapterekhez készítették el

Részletek:

<http://www.xen.org/files/xensummitboston08/Xen-SR-IOV.pdf>

<http://blog.scottlowe.org/2009/12/02/what-is-sr-iov/>

Teljes periféria emuláció



22



Teljes periféria emulációnál a virtualizációs környezetben vannak a fizikai eszköz meghajtó programjai (driver), minden perifériát ilyenek kezelnek.

A virtuális gép számára egy szoftveres „backend” (vagy „anti-driver”, driver ellendarab) emulálja a hardvert.

3 jellegzetes eset lehetséges:

- Valamilyen többszörös hozzáférést biztosító erőforrásmegosztó vagy ütemező komponens osztja szét a backendek felé a fizikai hardver erőforrást (Háttértár, hálózat, stb.)
- Statikusan valamelyik virtuális géphez hozzárendelünk egy-egy hardvert (Soros port, USB eszközök, esetleg háttértár, CD meghajtó)
- Távoli elérés szerver komponens hálózaton keresztül tesz elérhetővé valamilyen elemet a virtuális hardverből (Grafikus konzol, CD meghajtó)

Hardveres periféria virtualizáció

The diagram illustrates hardware peripheral virtualization through a layered architecture. At the top, a grey box represents the operating system, containing two orange speech bubbles. The left bubble states: "Lehet olyan hardveres periféria, amely közvetlenül a hardverrel kommunikál műveletekkel vagy adatokkal, egyszerre több virtuális gépből is. (Számos esetben)." The right bubble states: "Réteges architektúrájú busz elérési protokollok esetén (SCSI, USB) lehetséges: a busz adapter emulált rendszeren keresztül érhető el, de a magasabb rétegekben már a virtuális gép közvetlenül a hardverrel beszél." To the right of the OS box, a pink box labeled "Gépjármű" (Vehicle) has a red arrow pointing down to a stack of pink server racks. The word "Hardver" (Hardware) is written below the server racks. On the left, a green circuit board (representing a bus adapter) is connected via an orange cable to a grey "HDD" drive. A red arrow points from the OS layer down to the hardware components.

Lehet olyan hardveres periféria, amely közvetlenül a hardverrel kommunikál műveletekkel vagy adatokkal, egyszerre több virtuális gépből is. (Számos esetben.)

Réteges architektúrájú busz elérési protokollok esetén (SCSI, USB) lehetséges: a busz adapter emulált rendszeren keresztül érhető el, de a magasabb rétegekben már a virtuális gép közvetlenül a hardverrel beszél

Gépjármű

Hardver

HDD

23

Lehet olyan ha
közvetlen
műveletekkel v
egyszerre töb
gépből is. (S

Réteges architektúrájú busz
elérési protokollok esetén (SCSI,
USB) lehetséges: a busz adapter
emulált rendszeren keresztül
érhető el, de a magasabb
rétegekben már a virtuális gép
közvetlenül a hardverrel beszél

Hardver

Összefoglaló

■ A virtualizáció alapjai I-II

○ CPU virtualizáció

- CPU utasításkészlet architektúra és szerepe
- A három alap virtualizációs megközelítés

○ Memória virtualizáció

- Virtuális memória az operációs rendszerekben
- Virtuális memória a platform virtualizációban
- Virtuális memóriakezelés speciális képességei: megosztás, késleltetett allokáció, memória-ballon

○ Perifériák virtualizációja

- Perifériák programozói felülete általában
- Periféria virtualizációs architektúrák