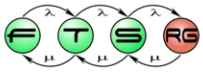
Virtualizációs technológiák és alkalmazásaik

Memória és perifériák virtualizációja

Micskei Zoltán, Tóth Dániel



Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

Utolsó módosítás:2011. 09. 29.

Tartalom

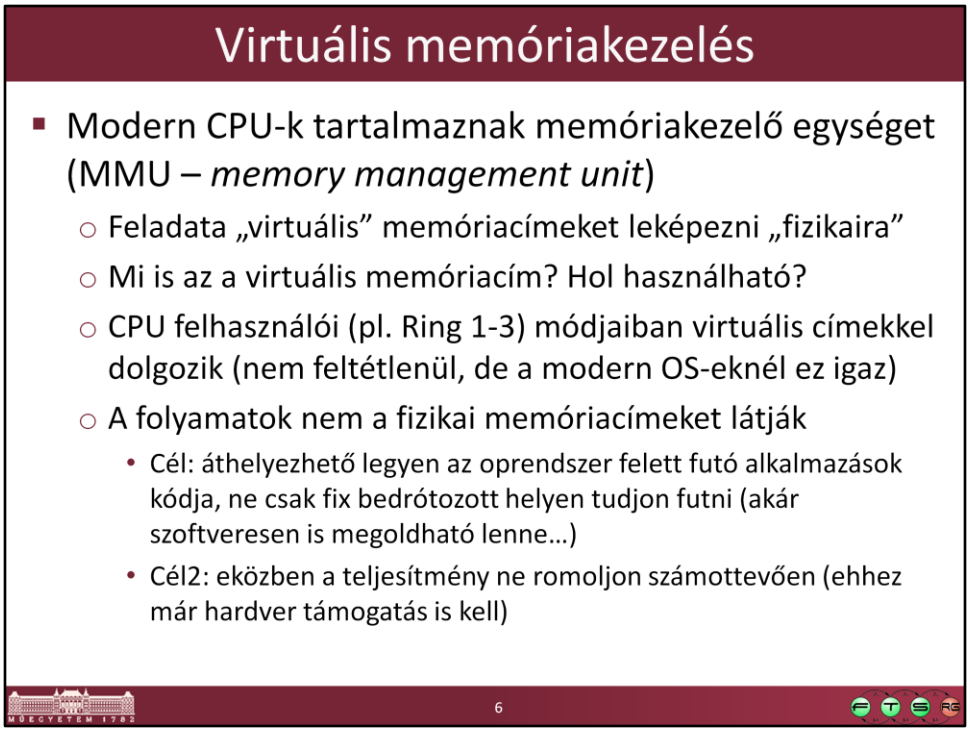
- Előző rész tartalmából:
  - CPU virtualizáció
  - A három alap virtualizációs megközelítés
- Memória virtualizáció
  - Virtuális memória az operációs rendszerekben
  - Virtuális memória a platform virtualizációban
  - Virtuális memóriakezelés speciális képességei: megosztás, késleltetett allokáció,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

Emlékeztető: 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

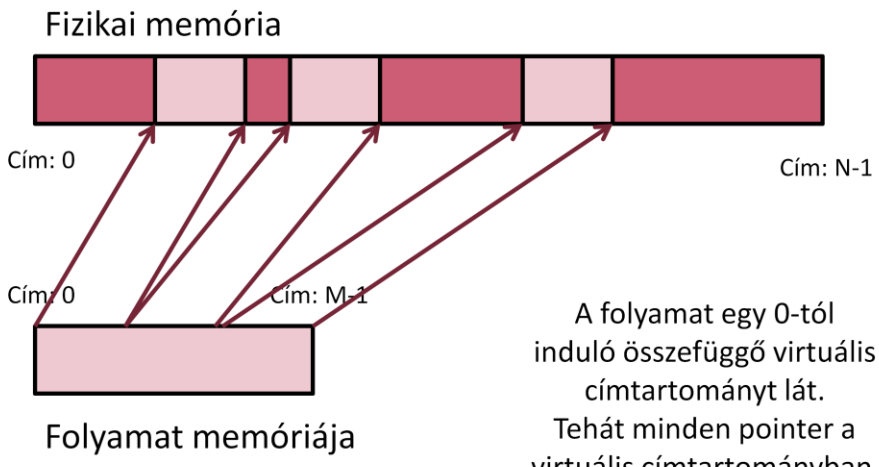


MMU != fizikai memóriaillesztő áramkör. Az utóbbinak a feladata a memória modulok elektromos alacsonyszintű vezérlése, ez sokáig a CPU-n kívül a chipset „északi hídban” foglalt helyet, csak a közelmúltban (Athlon64, Core i7) került be a processzorba. A memóriaillesztő áramkör is végez címfordítást, de 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.

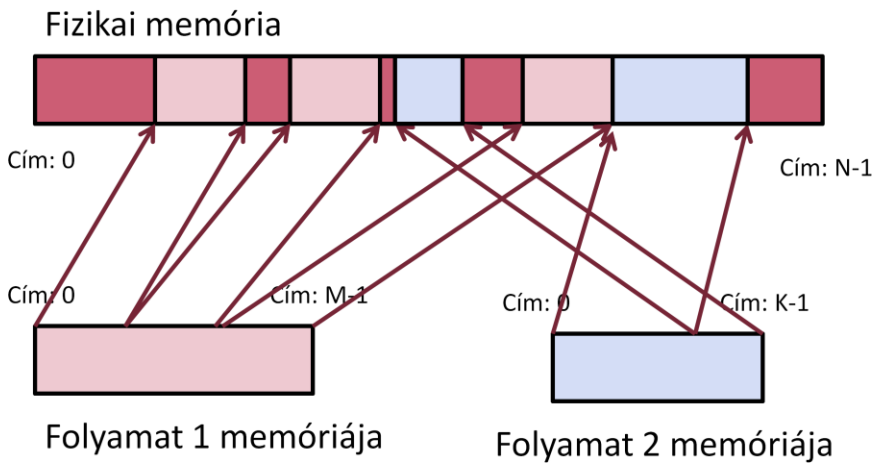
Részletek:
Intel® 64 and IA-32 Architectures Software Developer’s Manual
http://www.intel.com/products/processor/manuals/

Virtuális memóriakezelés



A folyamat egy 0-tól induló összefüggő virtuális címtartományt lát. Tehát minden pointer a virtuális címtartományban értelmezett.

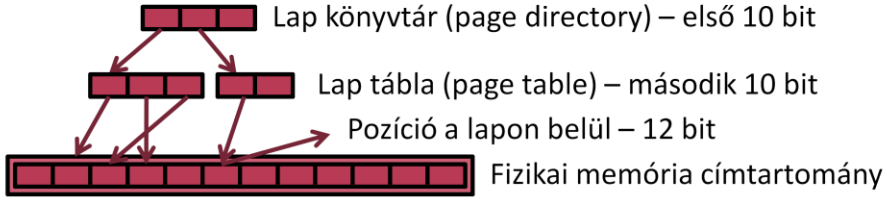
Virtuális memóriakezelés



Ez minden folyamatra 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



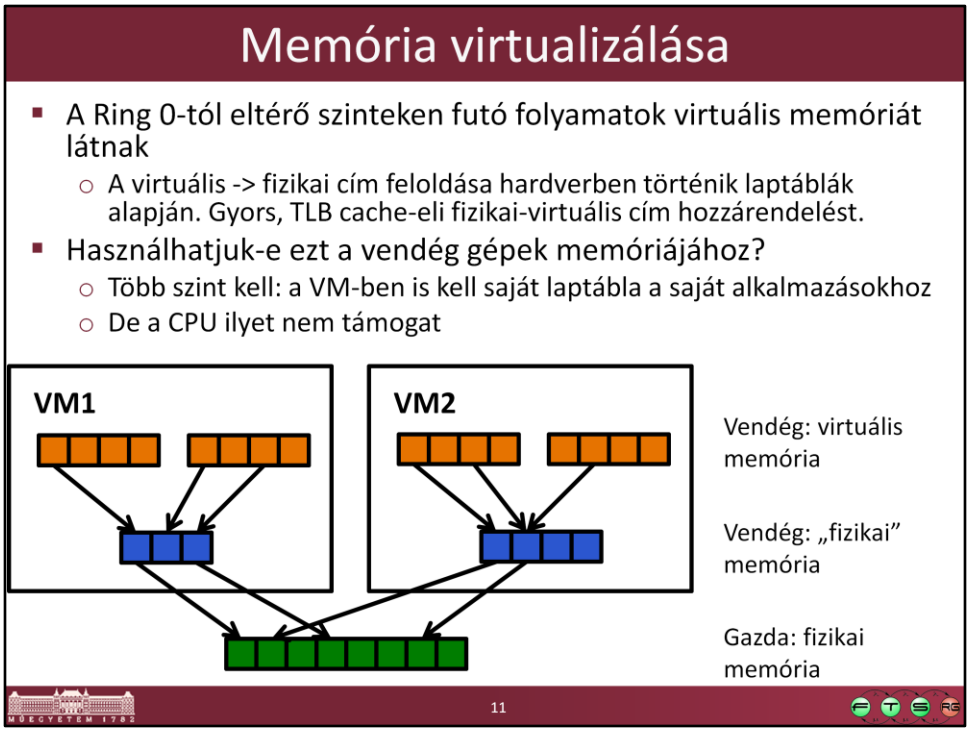
Megjegyzés: a felsorolt konkrét számadatok x86 architektúrán érvényesek, más architektúrán előfordul más beosztás is. Az x86-on is léteznek még speciális üzemmódok (PAE – paging area extension, NX – no execute), amik 1-1 felső helyiértékű bitet saját célra használnak.

Virtuális memória megvalósítása lapokkal

További jellegzetességek:

- A laptáblák is a fizikai memóriában foglalnak helyet
- Csak az operációs rendszer kernel módosíthatja őket
- Minden folyamathoz másik táblakészlet tartozik, a kernel kontextus váltáskor 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 hibát (fault) vált ki a CPU-ban

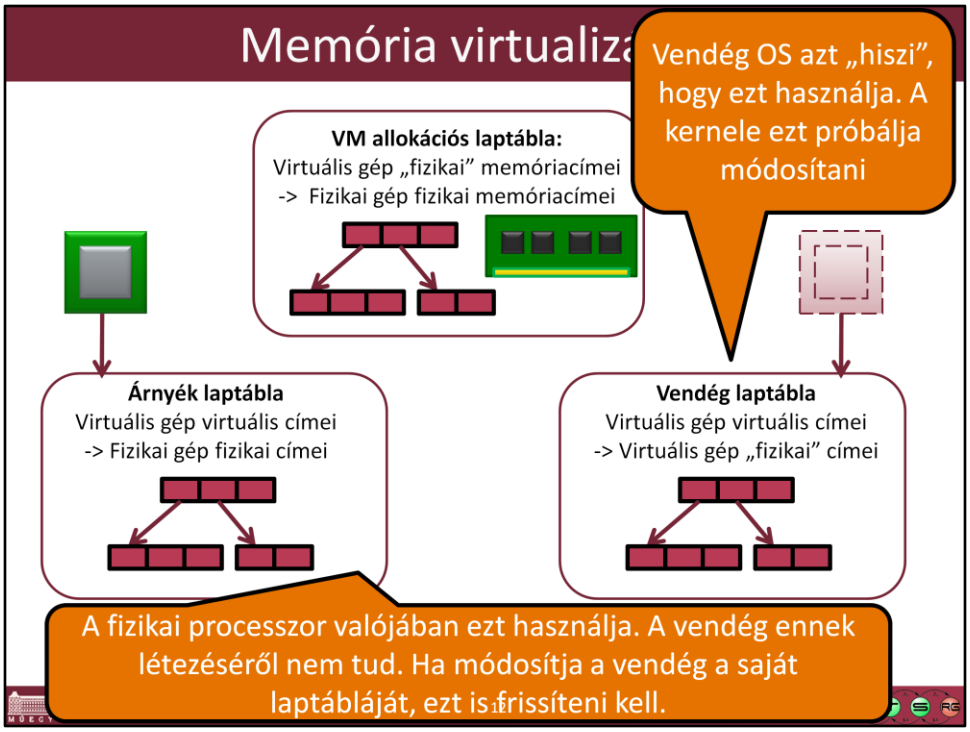
A **page fault** mechanizmus teszi lehetővé, hogy az éppen nem használt lapokat lemezre lehessen menteni majd vissza behúzni a memóriába akkor, amikor éppen hozzáférés történik. A hozzáférés a nem létező lapra fault-ot vált ki, ezt elfogja és ekkor teszi vissza a kernel a lapot a swap területről.



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 memóriatartományban, 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.



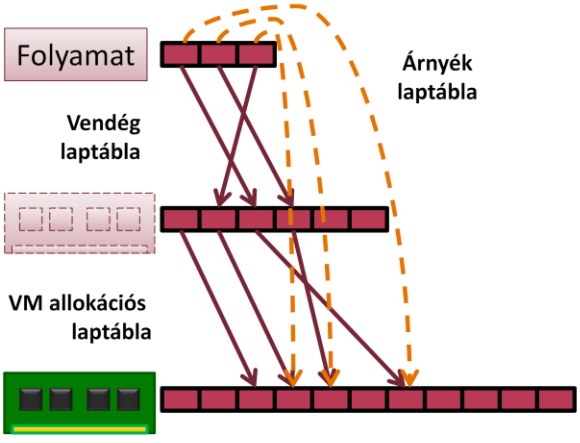
Kell két példány a laptáblából:

- az egyik a vendég kernel számára látható és a vendég VM „fizikai” címeit képezi le a VM-en belül futó folyamatok „virtuális” címeire,
- a másik a VMM (és a CPU) számára látható ún. *árnyék laptábla (Shadow page table),* ez a futtató gép „igazi fizikai” címeit képezi le a VM-en belül futó folyamatok virtuális címére.

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

Memória virtualizálása

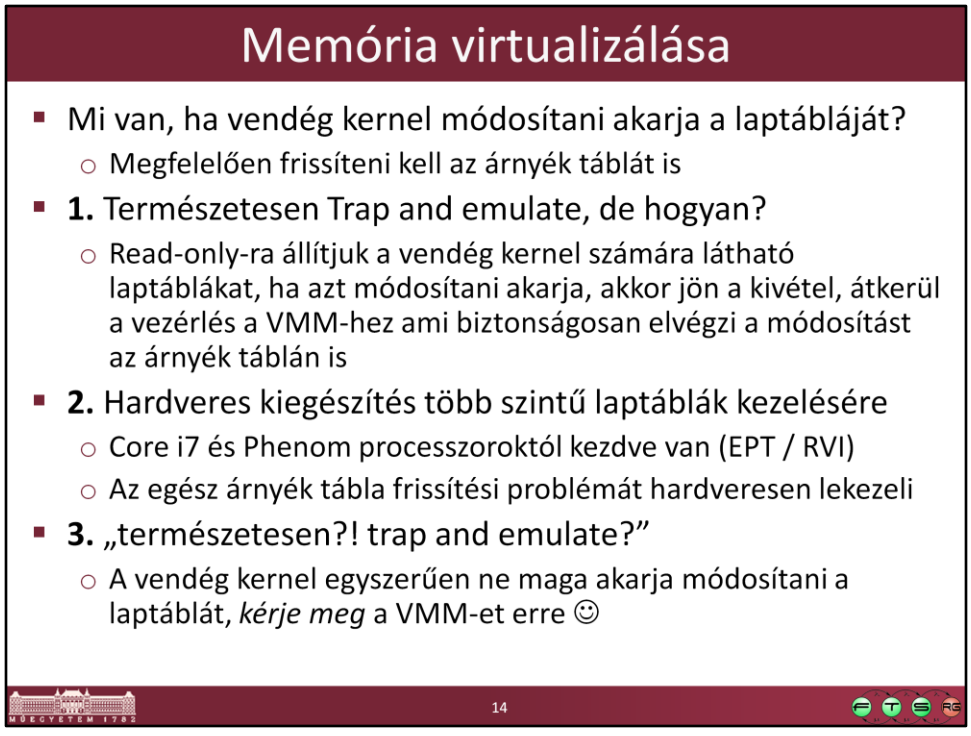
- Másik nézetben:



A virtuális gépben futó folyamat virtuális memóriája

A virtuális gép „fizikai” memóriája

Fizikai memória a hardverben



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.

Memória virtualizálása

Tanulságok:

- A memóriakezelésben is a háromféle fő megvalósítás megtalálható.
- A memóriakezeléshez extra laptáblák kellenek, tehát a VM több fizikai memóriát igényel, mint amennyi számára látható lesz
- A memória foglalása és felszabadítása extra feladatot jelenthet a VMM számára is (kivéve hardveresen virtualizált MMU-nál)

Extra memória virtualizálási lehetőségek

- Memórialap deduplikáció
  - azonos tartalmú memórialapok megosztása több vendég VM között
  - hasonlóképpen azonos lapok megosztása egy vendégen belül is
  - gyakorlati haszna főleg speciális alkalmazásokban (Virtual Desktop Infrastructure), tipikusan több példány fut azonos OS-ből
- Megvalósítása
  - gyors hash számítás, ez alapján egyezés keresés
  - közösített lapok megbontása beleíráskor, copy-on-write elv
- Hasonló: memória tömörítés
  - Egészen új lehetőség VMM-ekben (az ötlet persze régi)
  - CPU költsége nagyon nagy lenne, ezért:
  - az inaktív, amúgy háttértárra kilapozásra ítélt lapokat szokás tömöríteni -> a ki/be tömörítés még így is gyorsabb a merevlemeznél
  - Nem csodaszer… kompromisszumot kell kötni a tömörítetlen és tömörített lapoknak fenntartott memória mérete között, csak korlátozott méretben előnyös

Extra memória virtualizálási lehetőségek

- Dinamikus allokáció:
  - Ami memóriát nem használ a vendég, azt ne is kapja meg
  - Gyakorlati haszna önmagában elenyésző, a legtöbb OS az összes szabad memóriát disk cache-nek használja
  - Háttértárra swappelhetők a lapok a vendég OS tudta nélkül
- Memória felfújás (memory ballooning)
  - Ha kifogy a host memóriája, akkor „elvesz” a vendégtől
  - Egy ágens vagy driver a vendég kernelben (paravirtualizációs szemléletmód) elkezd memóriát foglalni a VMM utasítására. A VMM az ágens által „foglalt” 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

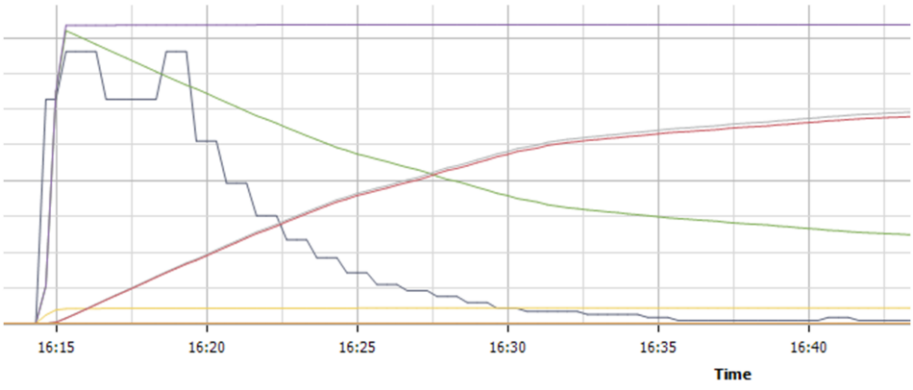
Miért olyan rossz az, ha a hoszt swappel? Mert a guest nem tudja, hogy a memóriatartományának egy része diszken van, tehát lassú, ezt a területet diszk cache-nek vagy alkalmazásoknak fogja használni, amivel még többet árt. Esetleg ha a host és a guest is swappel egyszerre, akkor lehet, hogy a guest a 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.

Extra memória virtualizálási lehetőségek

Tanulságok:

- Sokféle extra lehetőség, nem triviális megállapítani az aktuális használatot
- Figyelni kell a memóriafoglalást, nagyot esik a teljesítmény, ha lemezre kell lapozni
- Ha van rá lehetőség, ki kell használni a vendégbe telepíthető ballooning drivert, elkerülhető vele a vergődés

Kitekintés: VMware ESXi memóriakezelés



Jelmagyarázat:
granted (lila) active (kék) overhead (sárga) zero (piros) consumed (zöld) shared (szürkés-kék)

A képernyőképen az látható, hogy az indítás után a vendég gép memóriáját fokozatosan pásztázza végig a VMM, megosztható memórialapokat keresve, így fokozatosan emelkedik a shared memória mennyisége.

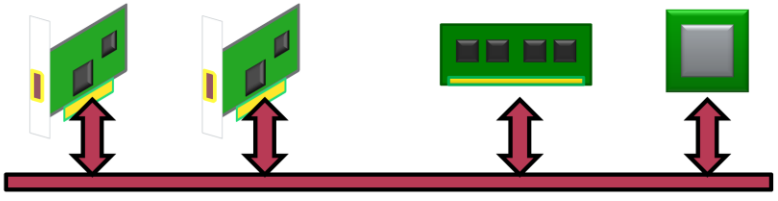
Tartalom

- 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

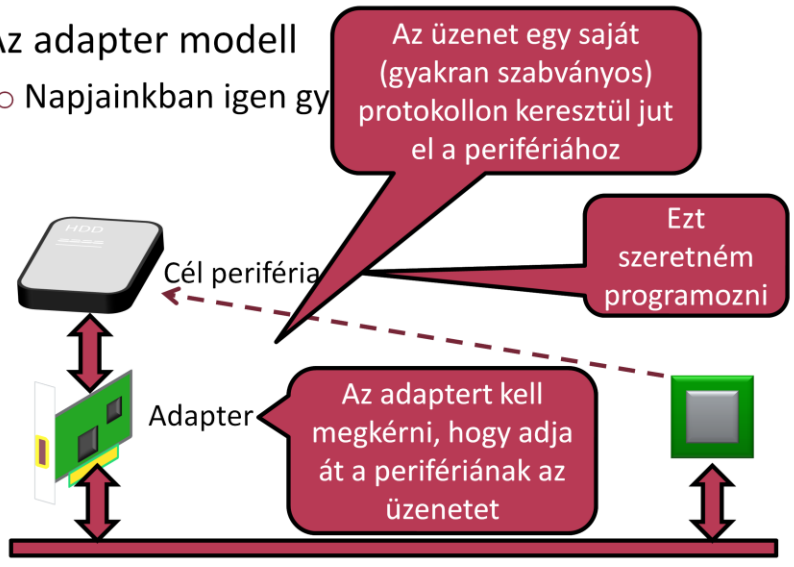
- A perifériák kezelése jellegzetesen
  - CPU felprogramozza a perifériát, regiszterek átírása
  - Periféria eseményt jelez a CPU felé, megszakítás
    - Ilyenkor valamilyen módon le kell kezelni az eseményt, valamit reagálni kell rá
  - Periféria maga elvégzi a feladatát, közvetlen memória hozzáférés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

A periféria kezelése a „driver” felelőssége



Perifériákról általában

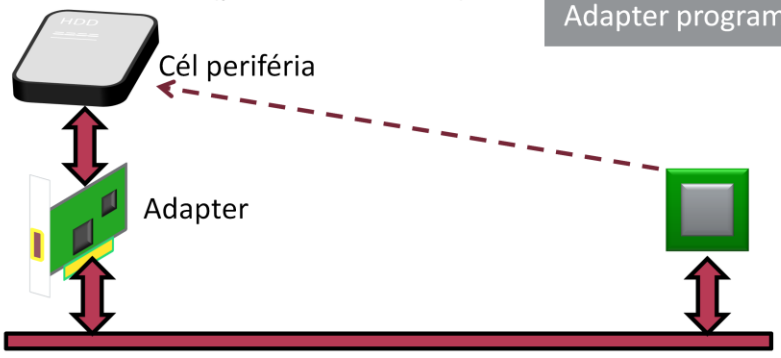
- Az adapter modell
  - Napjainkban igen gy



Perifériákról általában

- Az adapter modell
  - Napjainkban igen gyakori

Rétegzett programozási modell (pl. USB, SCSI, SATA):



Perifériának szánt konkrét utasítás, adat

Periféria protokollja

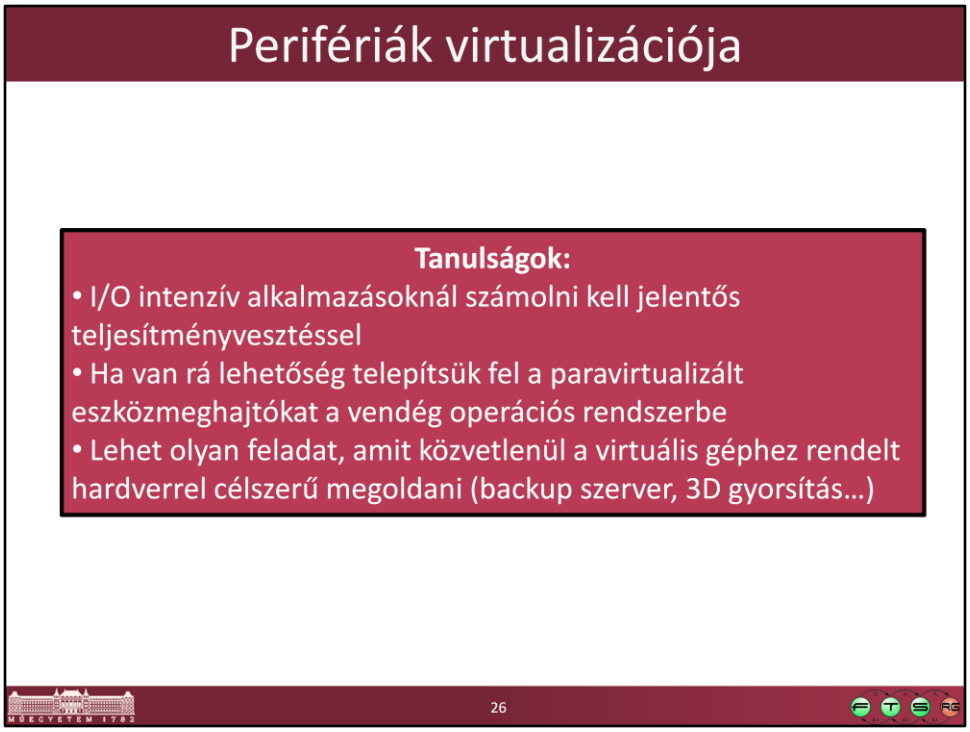
Adapter programozás

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 a VMM-ben -> lassú

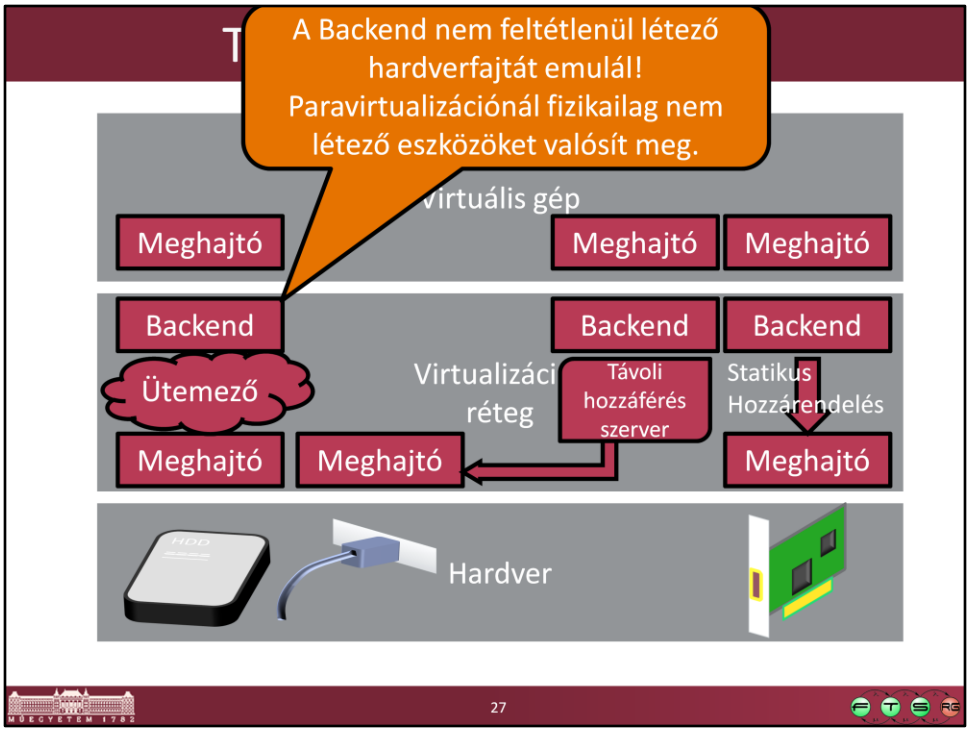
Perifériák virtualizációja

- 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
  - Saját „hardverek”, amik magas szintű műveleteket végeznek, pl. hoszt fájlrendszerhez hozzáférés. Itt kezd keveredni a virtualizáció az operációs rendszerekkel…
- Hardveres virtualizáció
  - Valamilyen hardvert közvetlenül elérhetővé teszünk a vendég számára
  - Tipikusan olyan esetekben működik, ahol a periféria valami emulált vezérlő adapter felett futó magasabb protokoll rétegben érhető el. Pl.: SCSI merevlemez, USB eszközök
  - Általános esetben veszélyes… DMA-val a fizikai memória egésze 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)



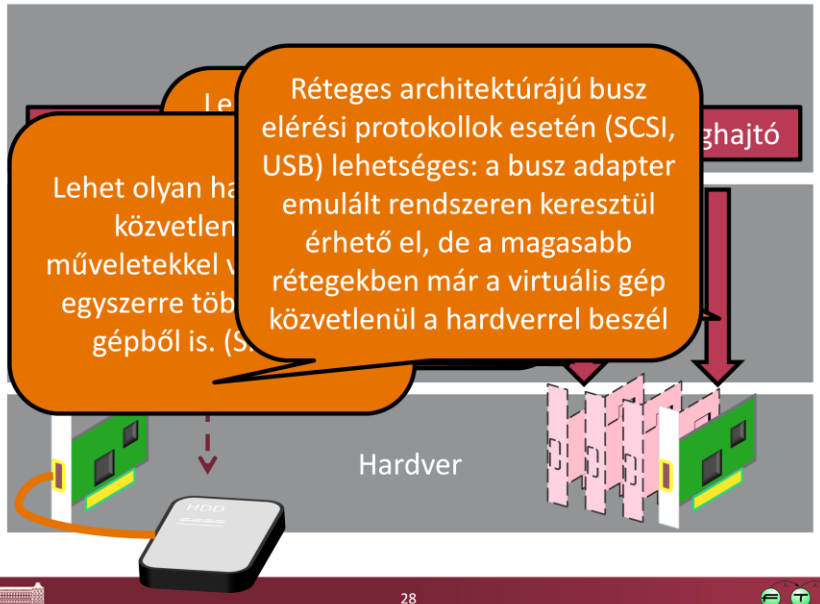
Fontos tanulság megint: amikor a paravirtualizációnál magas szintű erőforrásokat adunk át a guest OS-nek, akkor látható, hogy itt is - mint mindenhol máshol is – folytonos átmenet kezd kialakulni az operációs rendszerek és virtualizációs környezetek között. Tehát a szigorú taxonomizálás valójában megint 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 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 erőforrás-megosztó 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



További információ

- Darvas Dániel, Horányi Gergő. „Intel és AMD technológiák a hardveres virtualizáció megvalósítására”, virttech házi feladat, 2010.
- Abramson, D. *et al*. "Intel® Virtualization Technology for Directed I/O." Intel Technology Journal. (August 2006).
- Garaczi Tamás. Intel VT-d (IOMMU) technológia részleteinek megismerése, virttech HF, 2010.

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