Virtualizációs technológiák és alkalmazásaik (VIMIAV89)

Házi feladat:
Intel VT-d (IOMMU) technológia részleteinek megismerése

Garaczi Tamás
BIQYSD

2010.12.01.

I. Az I/O-eszközök virtualizációjának kihívásai

Bármely mai operációs rendszer egyik alapvető feladata a külvilággal való kommunikáció biztosítása – például a hálózati elérés, vagy akár a felhasználó felé nyújtott grafikus felület formájában. Ha egy számítógépen virtualizált rendszereket üzemeltetünk, ez a feladat bonyolultabbá válik, kiegészül két fontos funkcióval.

Az operációs rendszerek virtualizált futtatásának legfőbb feladata, hogy a fizikai hardver hagyományos korlátait átlépjük – ez jelen esetben azt jelenti, hogy egy adott (egyedi) IO-eszközt konkurrensen több virtualizált rendszerből elérhessünk.

Az IO-virtualizáció első feladata az eszköz elszigetelt elérésének biztosítása adott VM számára, vagyis megoldani azt, hogy a különböző VM-ek eszközkezelése ne zavarhassa meg egymást. Három követelménynek kell eleget tenni:

- Elszigeteltség: a szoftver és az IO-eszköz kommunikációja során az adott szoftverhez csak a neki szóló információk jussanak el.
- Megbízhatóság: az eszköz szolgáltatásai mindig, változatlan formában elérhetőek legyenek az adott szoftver számára
- Biztonság: a szoftver és az eszköz kommunikációjába kivülről, jogosulatlanul ne lehessen belehallgatni.

A második fontos feladat az erőforrások megosztása a VM-ek között, vagyis annak elkerülése, hogy minden VM számára külön eszközt kelljen telepíteni.

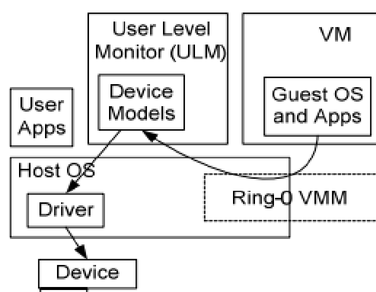
Az IO-virtualizáció minősége fontos tényező pl. adatközpontokban vagy virtualizált szerverekben, különösen a háttértárra és a hálózatra támaszkodó alkalmazások esetében, de kliens alkalmazások esetén is van létjogosultsága a technológiának: akár futtathatunk egy tűzfalat egy külön erre szánt VM-ben; ebben az esetben ennek a VM-nek kell megkapnia minden, a hálókártyáról érkező adatot, és miután megvizsgálta, továbbítani a többi VM felé.

II. Az egyes virtualizációs szoftver-architektúrák

Háromféle megoldást különböztetünk meg:

“OS-hosted” virtuálisgép-menedzserek[1. ábra]:

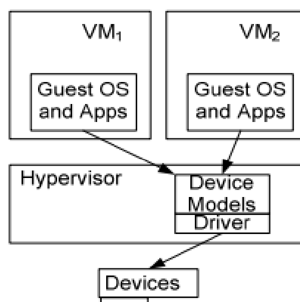
Az egyik lehetőség a VMM futtatására, hogy a korábban feltelepített operációs rendszer, a “gazdarendszer” fölé helyezzük, így a VMM a gazdagép szolgáltatásait fogja használni. Így a VMM egyes, eszközöket kezelő részei a gazdagép kernelébe épülnek, és átjárást biztosítanak a valódi hardver és a VM-eknek nyújtott virtuális eszközmodellek között, maga a virtuális gép pedig egy folyamatként fog futni a gazdagépen. Ennek előnye, hogy elméletileg bármilyen eszközt virtualizálhatunk bármekkora mennyiségben, hátránya viszont, hogy a gazdagép megbízhatósága (illetve megbízhatatlansága) rányomja bélyegét a VMM-re és az összes vendég gépre. Továbbá, mivel minden vendég OS egy-egy folyamat a gazdagépen belül, a gazda OS ütemezőjének hatáskörében lesznek, az viszont, mivel nincs tudatában annak hogy a folyamatok között teljes számítógépek lakoznak, nem tud megfelelő prioritást biztosítani nekik a saját folyamataival szemben – így ez a megoldás valós idejű követelményeknek sem felelhet meg.



1. ábra: OS-hosted megoldás

Hypervisor-ok[2. ábra]:

Egy másik lehetséges megközelítés a VMM egy réteggel lentebb helyezése – vagyis ne egy gazda OS kezelje és ütemezze a hardveres erőforrásokat, hanem maga a VMM. Így nincs is szükség gazda OS-re, az összes hardverközeli kód a VMM-be kerül – ezt hívjuk Hypervisor-nak. Ennek köszönhetően nem ütközünk bele az OS-hosted rendszerek problémáiba a megbízhatóság és a teljesítmény terén, viszont ez nagyban bonyolítja a VMM elkészítését, hiszen minden egyes használandó eszköztípusnak el kell készíteni a megfelelő meghajtóprogramot a VMM alá. Ezzel elveszítjük az OS-hosted megoldások rugalmasságát a felhasználható IO-eszközök tekintetében – a megfelelő működés a hordozhatóság rovására megy.

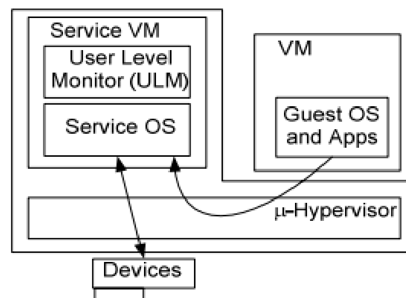


2. ábra: Hypervisor

Hibrid VMM-ek[3. ábra]:

A harmadik megközelítés a hosted megoldások és a hypervisor-ok előnyeit igyekszik egyesíteni. Gyakorlatilag megcseréli az OS-hosted architektúra Host OS és VMM rétegét. Ezzel a cserével egy olyan, kis méretű hypervisort kaphatunk, amely kezeli a processzor-és memória-erőforrásokat, az IO-eszközök felelősségét (driverek, ütemezés...) viszont átadja az általa (virtualizáltan) futtatott ún. Service OS-nek, melynek ez az egyetlen feladata. Így a meghajtók megírását és integrációját egyszerűbbé tehetjük, hiszen csak egy erre specializált mini operációs rendszerbe kell beilleszteni őket, és ezzel együtt megtarthatjuk a hypervisor adta előnyöket az ütemezés teljesítménye, illetve a megbízhatóság terén.

Ennek hátránya viszont, hogy a Service OS és a vendég OS-ek között folyamatos kontextus-váltásra van szükség az IO-eszközök használatakor, valamint meg kell oldani, hogy az eszközök DMA-hozzáférései a Service OS-hez érkezzenek be. Ez utóbbi nem egyértelmű, hiszen a Service OS nem áll közvetlen kapcsolatban a hardverrel, erre ad majd megoldást az Intel VT-d, illetve az AMD IOMMU technológiája.



3. ábra: Hibrid VMM

III. IO-virtualizációs technikák

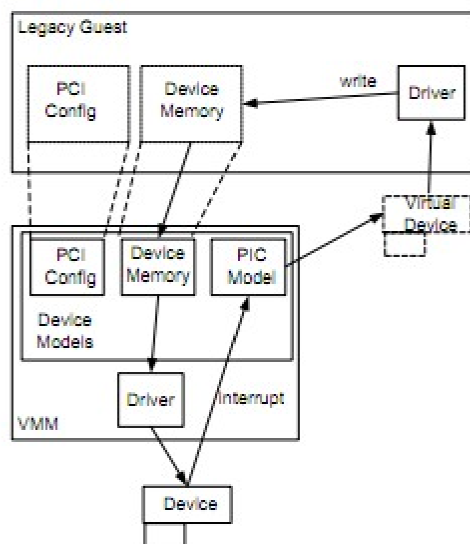
Milyen feladatokat kell megoldani egy IO-eszköz kezelésekor?

- Az eszköz felfedezhetőségének biztosítása az operációs rendszer számára,
- az eszköz vezérlése,
- az eszköz és az operációs rendszer közötti adatátvitel biztosítása (tipikusan DMA-hozzáférések),
- az eszköztől érkező megszakítások kezelése.

Ezen feladatokra több megoldás is létezik:

Emuláció[4. ábra]:

Egy létező eszközt emulálunk mind a négy funkciójával, teljesen szoftveresen, a VMM-ben. Ennek nagyon nagy előnye a migráció korlátlansága, vagyis bármilyen másik hardverre áthelyezhetjük a VM-ünket, az ebből nem fog érzékelni semmit. Továbbá lehetőségünk van az emulált eszközből több példányt is létrehozni, “megosztani” az eszközt; és ez a megoldás teljesen OS-független, hiszen az emuláció a VMM-ben történik, a VM-ek tudta nélkül.



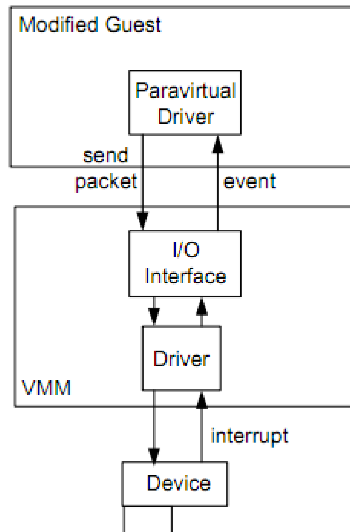
4. ábra: Eszköz emulációja

Az emulációnak azonban ára van, ami a teljesítményben mutatkozik meg. Egy eszköz teljesen szoftveres emulációja hatalmas plusz terhet ró a CPU-ra, ráadásul mivel egy létező eszközt szeretnénk mutatni a vendég OS felé, annak minden ismert hibáját is emulálnunk kell, hiszen a vendég meghajtóiba már be van építve a hibák kezelése is – ez tovább bonyolítja az emulációt.

Paravirtualizáció[5. ábra]:

Egy újabb lehetőség az IO-eszközök virtualizációjára, hogy létrehozunk egy magasabb szintű interfészt az OS és a hardver közti kommunikációra. A VMM-ben létrehozunk egy réteget ami elvégzi a hardverközeli és a paravirtualizált interfész közti fordítást, a vendég OS-t pedig módosítjuk, hogy hardver helyett a VMM-ben lévő paravirtualizált interfésszel kommunikáljon – a hardver megszakításait pedig a VMM események formájában jelzi a vendégnek.

Így megszabadulunk az emuláció plusz számításigényeitől, viszont elveszítjük az OS-függetlenséget, hiszen a vendéget módosítani kell hogy tudjon róla hogy virtualizáltan fut, és a módosítás OS-specifikus elemekből kell hogy álljon.



5. ábra: Paravirtualizált hozzáférés

Közvetlen hozzárendelés:

A legjobb megoldás az lenne az eszközök virtualizációjára, ha a vendég OS-ek közvetlenül a fizikai hardverrel tarthatnák a kapcsolatot – mind teljesítmény, mind a kommunikáció egyszerűsödése (és így a kevesebb hibalehetőség) szempontjából.

Azonban ez egyrészt felvet migrációs problémákat – hogyan vigyük át hatékonyan a VM-et egy másik hardverre, ha a vendég OS a régi hardver kezelésére van felkészülve?

A másik problémakör az IO-virtualizáció két alapfeladatával kapcsolatos – az elszigetelt, biztonságos, megbízható hozzáférés és az eszközmegosztás problémája.

Utóbbira jelenleg még nincs hardveres megoldás, előbbi viszont már megoldott, a kérdés csupán az: hogyan mondjuk meg a hardvernek, hogy melyik vendég operációs rendszerrel próbáljon meg kommunikálni?

Ezt a problémát oldja meg hardveres eszközzel az Intel VT-d, illetve az AMD IOMMU.

A közvetlen hozzárendelést az IOVM-architektúrában használják leginkább elterjedten, melynek lényege, hogy egy adott eszköz közvetlen, elszigetelt kezelésére egy mini VM-et hozunk létre, majd ez a VM fogja a többivel megosztani az eszközt, akár emuláció, akár paravirtualizáció útján.

IV. Az IO-virtualizáció hardveres támogatása

Az IO-eszközök kezelésének részfeladataiból az első kettő, a felfedezhetőség és a vezérlés virtualizációjára a jelenleg létező szoftveres (CPU virtualizációs) eszközök is kielégítő eredményt adnak. A hardverben megoldandó feladatok a DMA-hozzáférések és a megszakítások virtualizációja.

DMA címfordítás

A VT-d architektúra ahhoz, hogy elszigetelje az egyes VM-ekhez tartozó DMA-hozzáféréseket, ún. protection domain-ekre osztja az elérhető memóriát. Egy VM-hez egy protection domain tartozik, de egy domain-hez természetesen több eszköz is hozzáférhet megfelelő jogosultság birtokában.

A "jogosultságokat", vagyis a hozzáférés szabályozását a VT-d címfordítással valósítja meg. Azért van szükség címfordításra, mert a DMA-t kérő eszköz a teljes fizikai memóriát, mint homogén címterületet látja, és ezen belül próbál használni hagyományosan egy IO-címtartományt, míg a virtualizált környezetben minden egyes VM saját doménjében van egy-egy IO-tartomány, amit csak az adott vendég OS használ. A két címzésforma közti átjárást a VT-d hierarchikus címfordító táblák segítségével biztosítja, melyek fizikailag az északi hídon helyezkednek el.

A címfordítás menete[7. ábra]

A fordítás logikailag két részre osztható, az első az adott eszköz és a hozzá tartozó protection domain összerendelése, a második pedig ez alapján a hozzárendelés alapján a konkrét címfordítás elvégzése.

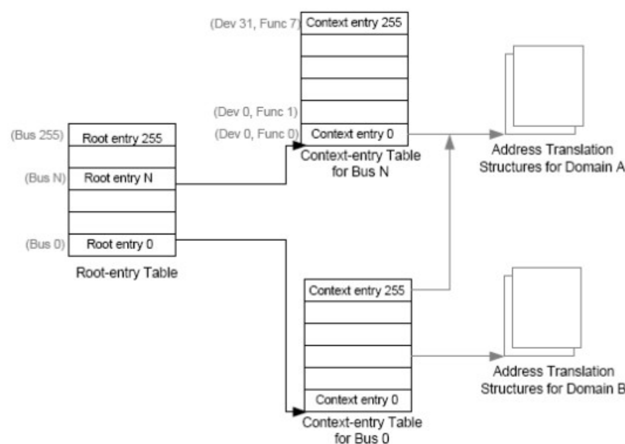
Az IO-eszköz által használt általános IO-cím és az adott protection domainhez tartozó cím közti átjáráshoz a VT-d-nek azonosítania kell a DMA-kérés küldőjét. Ehhez a PCI szabvány adja kezünkbe az eszközt: minden egyes DMA-kérés hordozza a küldője azonosítóját, a következő összetételben:

- busz azonosító,
- eszköz azonosító,
- funkció azonosító.

Ez alapján a címfordításhoz használt táblák az alábbi hierarchiába szerveződnek:

- Root Entry Table: kulcsa a busz azonosító; a tábla egy sora megadja egy adott buszon levő eszközökhöz tartozó Context-Entry Table címét.
- Context Entry Table: kulcsa az eszköz+funkció azonosító; a tábla megadja a kérést küldő eszköz számára elérhető protection domain címfordító struktúrájának címét.

Az egyes protection domain-ekhez tartozó címfordító struktúrák felépítése pedig hasonló a hagyományos, MMU-kban használt többszintű laptábláékéval.



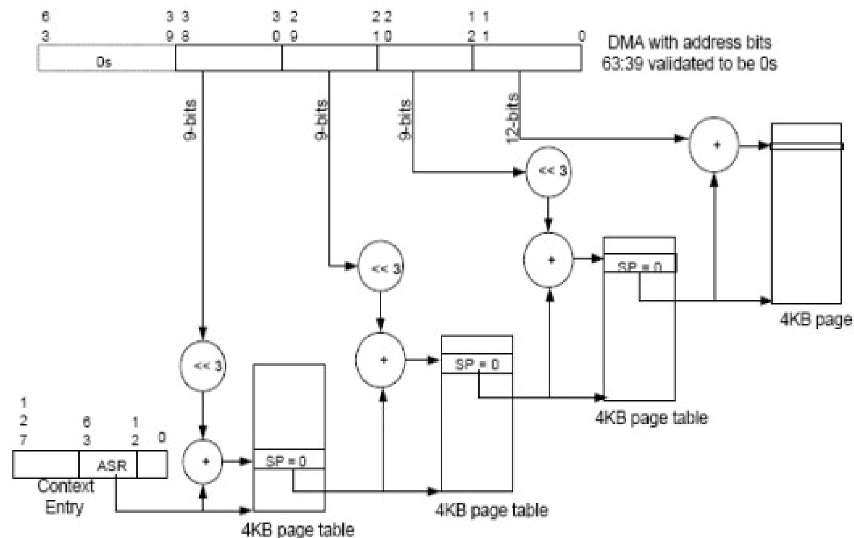
7. ábra: Az eszköz-azonosítók és a protection domain-ek összerendelése

Megszakítások címfordítása[7. ábra]

Az izoláció biztosításához a VMM-nek a megszakításokat is kezelnie kell, a DMA-kérésekhez hasonlóan. Alapvetően kétféle megszakítás-kérés lehetséges: az I/O-megszakítás-kezelőkön keresztül küldött kérések, illetve a DMA-ba ágyazott megszakítás-kérések. Ez utóbbiak egy meghatározott címre érkeznek, és a címben, illetve a hordozott üzenetben vannak kódolva a megszakítás tulajdonságai.

A megszakítások izolálását a VT-d úgy oldja meg, hogy az üzenet helyére egy megszakítás-azonostót tesz, amihez aztán egyrészt az egyszerű DMA-kérésekhez hasonlóan hozzárendelhető a protection domain, másrészt utólag hozzárendelhetők a megszakítás tulajdonságai egy előre felépített táblából.

A VT-d architektúra mindkét típusú megszakítás átirányítására képes; a DMA- és megszakítás-kérések átirányításának képességét egyébként a megfelelően megírt OS-ek saját teljesítményük és megbízhatóságuk javítására is hasznosíthatják.



7. ábra: Példa egy 3-szintű laptáblára

Gyorsítótárak

A címfordítások sebességének növelésének érdekében a VT-d tartalmaz hardverelemeket a fordítási táblák gyorsítótárazásához is:

- Context Cache: A Context Entry Table gyakran használt elemeit tárolja, a protection domain-ek gyors eléréséhez.
- Page Directory Entry Table: A címfordító laptáblák bejárása során leggyakrabban feljegyzett elemeket tárolja.
- IO TLB (Translation Look-aside Buffer): A gyakori címfordítások végeredményeit tárolja.
- Interrupt Entry Table: A megszakítások címfordításához leggyakrabban használt táblabejegyzéseket tartalmazza.
- Bár az egyes gyorsítótárakat a hardver kezeli, a konzisztencia fenntartásáért, az elévült bejegyzések invalidációjáért a szoftvernek kell felelnie. Erre két lehetőség adott:
- Synchronous Invalidation: A szoftver meghatározott memóriacímek használatával kérhet invalidációt, illetve vizsgálhatja, hogy a kérései végrehajtódtak-e.
- Queued Invalidation: A szoftver kérései bekerülnek a memóriában tárolt várakozósorba. A szinkronizációt a szoftver egy wait paranccsal kérheti: a hardver ennek sikerességét jelezni fogja, ha a parancs beérkezéséig kapott összes invalidáció-kérést kiszolgálta.

A gyorsítótárak használatával a címfordítások során nagyban csökkenthető a memóriában lévő táblákhoz való hozzáférések költsége, viszont a gyorsítótárak skálázása egy újabb kihívás elé állította a tervezőket.

Míg a processzorban elhelyezkedő TLB-nek hagyományosan csak egy processzor kéréseit kell szekvenciálisan kiszolgálnia, az IOTLB-knek több eszköz konkurrens hozzáférésével kell számolnia.

Ennek egy megoldása, ha maguk az IO-eszközök is részt vesznek a címfordítások gyorsítótárazásában azzal, hogy a saját kéréseikhez tartozó fordítási eredményeket helyben, a saját IOTLB-ikben tárolják.

Ennek megvalósítását célozza meg a PCI-Express szabvány Address Translation Services nevű kiegészítése. Az ATS protokoll lehetővé teszi, hogy egy IO-eszköz címfordítást kérjen a VT-d hardvertől, a központi címfordító hardver pedig visszatérítse a fordítás eredményét az adott eszköznek, hogy az eltárolhassa a saját IOTLB-jében. Ezután, ha az eszköz egy elküldendő DMA-kérés fordítási eredményét megtalálja a saját tárolójában, maga is elvégezheti a fordítást, és jelezheti ezt a központi hardvernek, így annak már nem kell foglalkoznia a kereséssel, csak átengedni a kérést. Emellett az is megoldott, hogy a szoftver a központi VT-d hardveren keresztül karban tarthassa a kihelyezett gyorsítótárak tartalmát; illetve, hogy a külső tárakba csak a központi adatstruktúrából származó, érvényes fordítási eredmények kerülhessenek eltárolásra.

Hibakezelés

A címfordítások során fellépő hibákat, jogosulatlan hozzáféréseket a hardver címfordítási laphibaként érzékeli. A laphibát okozó DMA-kérések blokkolásra kerülnek, és olvasási kérés esetén a hiba válaszban kerül jelzésre az eszköz számára. A hardver naplózza a laphibát okozó kéréseket, és események formájában értesíti róluk a szoftvert.

A kifejezetten címfordítást kérő eszközök (pl. saját IOTLB-jük frissítése céljából), ha kérésükkel hibát okoznak, azt a hardver nem értelmezi laphibaként, hanem egyszerűen értesíti az adott eszközt a hibás fordításról. Így az alkalmas eszközök a saját laphiba-kezelésük feladatát is átvehetik a központi DMA-hardvertől.

V. Összefoglalás

A jelenlegi technológiák egészen hatékony hardveres megoldást adnak az IO-eszközök direkt hozzárendelésére egy adott VM-hez, a megosztás biztosítása viszont még a jövő feladata. A VT-d segítségével egy eszközt minőségében megoszthatunk, azaz az általa képviselt funkciókat elszigetelten, akár különböző VM-ekhez is hozzárendelhetjük, mennyiségi megosztásra viszont ez a technológia nem alkalmas – erre a feladatra továbbra is marad a paravirtualizáció, illetve az emuláció.

Forrás:

http://download.intel.com/technology/itj/2006/v10i3/v10-i3-art02.pdf
Intel Virtualization Technology for Directed I/O