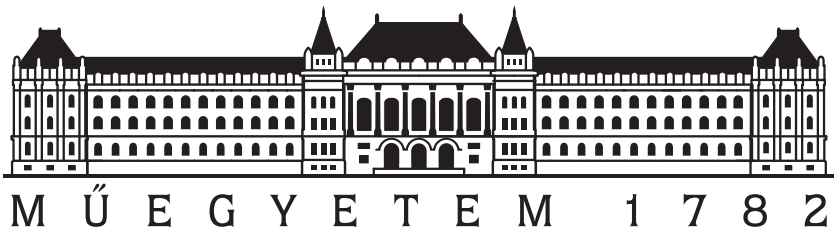


Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

Intel és AMD technológiák a hardveres virtualizáció megvalósítására

Virtualizációs technológiák és alkalmazásaik (VIMIAV89)

Házi feladat

Készítette:
Darvas Dániel
Horányi Gergő

2010.

Tartalomjegyzék

1. Bevezető **2**
1.1. Virtualizáció típusai 2
1.2. Történelmi áttekintés 3

2. Processzorok virtualizációs technológiái **4**
2.1. Root mode 4
2.1.1. Guest mód használata AMD processzorokban 4
2.1.2. Az Intel VMX 6
2.2. Címfeloldás támogatása 7
2.3. TLB-t érintő technológiák 9
2.4. Migráció támogatása 10
2.5. Biztonsági technológiák 11
2.6. FlexPriority 12
2.7. Egyéb technológiák 12

3. Összefoglalás **15**

1. Bevezető

A virtualizáció motivációja kezdetektől fogva a számítógépek jobb kihasználása. Ha egy szerverre egyetlen alkalmazást telepítünk, tipikusan nem használjuk ki a rendelkezésre álló erőforrásokat. Amennyiben viszont egy szerverre több alkalmazást is telepítünk, az biztonsági, kompatibilitási és karbantartási problémákat vet fel. A virtualizáció egy megoldást nyújt erre, mivel egy fizikai számítógépen több virtuális számítógép is használható úgy, hogy megvalósul az egyes alkalmazások elszeparálása, viszont a fizikai gép kihasználása is növekszik, emellett az üzemeltetés, karbantartás is könnyebbé válhat. A virtualizálásnak kliens oldalon is van tere, egymással inkompatiblis alkalmazások használatára vagy eltérő operációs rendszerek egyidejű futtatására ad megoldást egy fizikai gépen.

Ahhoz, hogy a virtualizálás x86 platformon elterjedten használatos legyen, számos problémát kellett leküzdeni. Ezek közül egy, hogy a processzorok nem voltak felkészítve több operációs rendszer egyidejű futtatására. Ez biztonsági és jelentős teljesítménybeli problémákat egyaránt okozott. Gondot jelentett az x86 platform virtualizálása során többek közt a virtuális gépek memóriájának fizikai memóriára képezése, a vendég gépek hardverhozzáférése, a memória védeleme, a DMA kezelése [20].

A hardveres virtualizációs támogatás fejlődése azonban lehetővé tette, hogy a virtualizálás ne csak a tesztelés és fejlesztés során jelenjen meg, hanem éles, missziókritikus vállalati alkalmazások esetén is használhatóvá és elterjedté váljon [5]. A következőkben a processzorokba épített virtualizációs támogatás egyes elemeit tekintjük át az Intel VT-x és az AMD-V technológiákban. Ezek mellett más hardveres támogatások is léteznek, melyek nem képezik dolgozatunk tárgyát. Ilyen az IO funkciók virtualizálását segítő Intel VT-d és AMD-Vi (IOMMU) vagy az Intel Virtualization Technology for Connectivity (VT-c).

1.1. Virtualizáció típusai

Ahhoz, hogy a hardveres virtualizációt megfelelő kontextusban lehessen kezelni, szükséges az egyéb virtualizációs típusok nagyon rövid áttekintése. Ezek abban térnek el egymástól, hogy az x86 architektúra problémás, privilegizált utasításait hogyan hajtják végre.

Paravirtualizáció Paravirtualizáció esetén a vendég rendszert úgy módosítják, hogy a virtuális gép által kiadandó, problémákba ütköző utasításokat speciális utasításokra cserélik, melyek futása már akadálytalanul lezajlik. Előnye, hogy nem szükséges emulálni a vendég operációs rendszer számára az utasítások végrehajtását, viszont az operációs rendszer kernelébe történő beavatkozás nehézkes és sok esetben jogi korlátokba is ütközik. Emellett a későbbiekben bemutatásra kerülő ekvivalencia kritériumot sem sikerül teljesíteni, mivel a vendég operációs rendszer más működésű lesz, mint egy fizikai gépen futtatva.

Szoftveres virtualizáció A *bináris fordítás* (Binary Translation) vagy más néven szoftveres virtualizáció során a vendég operációs rendszer kódja változatlan marad. Amennyiben egy olyan utasítás hívódik, amelyhez a vendég operációs rendszernek nem volt joga (*privilegizált utasítások*), akkor a processzorban egy belső hardveres megszakítás váltódik ki (*trap*). Ezen keresztül értesülhet a virtualizációs szoftver a problémás utasításokról és emulálhatja annak hatását. Azonban bizonyos utasítások nem váltanak ki trap-et (ez az x86-os architektúra egy komoly problémája a virtualizáció szemszögéből), így ezeket futás közben, *just in time* kell kicserélni.

Hardveres virtualizáció Ebben az esetben a processzor hardveresen támogatja a virtualizációt, így a kiegészítései segítségével elegáns megoldást ad azokra a hibákra, melyeket az előbbi megoldások próbáltak kiküszöbölni. Azonban nem jelenthető ki egyértelműen, hogy ez lenne a legjobb módszer, bár egyre jobb eredményeket ér el. Munkánk során az ezt támogató technológiákat mutatjuk be.

1.2. Történelmi áttekintés

1974-ben Gerald J. Popek és Robert P. Goldberg leírta [14], hogy milyen követelményeknek kell megfelelnie egy virtualizációt megvalósító szoftvernek (*virtual machine monitor*, VMM). Ezek a következők:

Ekvivalencia. Egy virtuális gép számára a fizikaival teljesen azonos környezetet kell biztosítani.

Biztonság. A VMM-nek a virtualizált erőforrások felett teljes rendelkezéssel kell bírnia.

Teljesítmény. Az utasítások nagy részének a VMM beavatkozása nélkül kell futnia.

Ezeknek a követelményeknek a személyi számítógépekben elterjedt x86 architektúra eredetileg nem felelt meg. A bináris fordítás (másképp szoftveres virtualizáció) esetében, mellyel az ekvivalencia kritériuma megvalósítható volt, a teljesítmény kritériumának biztosítása nehéz és összetett probléma. Emellett bizonyos utasítások esetén a biztonság kritériuma nem teljesült. A paravirtualizációhoz pedig beavatkozás szükséges a virtuális gép operációs rendszerébe, amely sértheti az ekvivalencia kritériumát, illetve jogi problémákat is felvet. Az AMD és az Intel nagyjából egyszerre, 2005-tól kezdődően, egymástól függetlenül kidolgozott új kiegészítéseket processzoraikhoz. Ezen hardveres támogatás segítségével az x86 architektúra mára jelentős teljesítménycsökkenés nélkül virtualizálhatóvá vált [16].

2. Processzorok virtualizációs technológiái

Ebben a fejezetben bemutatjuk az Intel VT-x és az AMD-V fontosabb elemeit. Ebben a dolgozatban azonban nincs lehetőség az összes technológia kimerítő bemutatására. Bővebben az AMD-V-ről a [2][6]-ban, az Intel VT-x-ről a [12][16]-ban olvashat.

2.1. Root mode

Az x86 platformon különböző privilégiumszintek értelmezettek, ezeket *ring*-eknek nevezik. A legfrissebb, virtualizációs támogatások nélküli processzorokban 4 ringet használtak (illetve használnak még mindig, mivel továbbra sem tartalmaz minden processzor hardveres virtualizációs támogatást). A legmagasabb szintű jogokkal rendelkező Ring 0-ban fut az operációs rendszer, amely így teljes hozzáféréssel rendelkezik. Azonban nem futhat Ring 0-ban a gazda és a vendég operációs rendszer egyszerre. Amennyiben a vendég operációs rendszer egy magasabb számú (azaz alacsonyabb jogú) ringben fut, nincs joga bizonyos szükséges műveletek elvégzésére. Ezeket a műveleteket vagy át kell írni az operációs rendszer kódjában és adaptálni kell a magasabb ringben futáshoz (ekkor paravirtualizálásról beszélünk) vagy szoftveres virtualizáció esetén emulálni kell ezeket a funkciókat, amely overheadet jelent [20].

Ennek elkerülésére bevezettek egy *root* és egy *guest* módot a processzorokban, amelyek segítségével a vendég operációs rendszerek a megszokott Ring 0-ban, guest módban futhatnak, míg a virtualizációt vezérlő rendszer (*Virtual Machine Monitor*) pedig a root mód Ring 0-ban futhat. A két mód (azaz a VMM és a virtuális gép kontextusa) közötti váltást *world switch*-nek nevezzük. Így lehetséges az, hogy a vendég rendszer módosítás nélkül futhasson, azonban mégis csökkentett jogokkal, míg a virtualizációért felelős szoftverrendszer a teljes rendszert vezérelheti. Ezt a felépítést mutatja az 1. ábra.

1. ábra. Hardveres virtualizáció esetén létező CPU Ring-ek (forrás: [18])

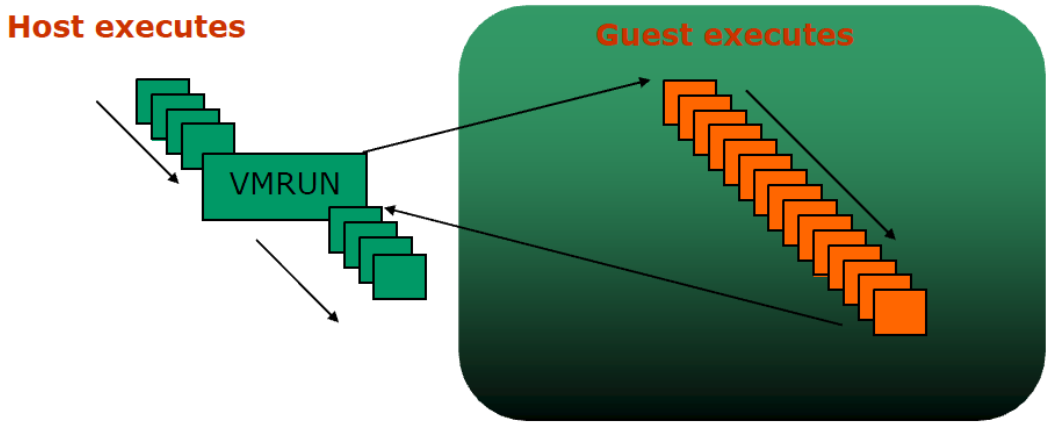
2.1.1. Guest mód használata AMD processzorokban

A world switch lebonyolítására az AMD processzorokba néhány új utasítás került, ezek közül a fontosabbak: VMRUN, VMSAVE, VMLOAD, VMMCALL. Egy fontos új kivétel (*exception*) is van, ez a #VMEXIT.

A VMM és a virtuális gép kontextusa közötti váltás tömören összefoglalva a következőképp zajlik:

- A processzor a VMM által kiadott VMRUN utasítás hatására guest módba lép.
- Helyreállításra kerül a vendég gép kontextusa.
- A vendég gép utasításai futnak.
- Egy #VMEXIT kivétel hatására a vendég gép futása megszakad.
- Helyreállításra kerül a gazdagép kontextusa.
- A VMM kódjának futása folytatódik a VMRUN utáni utasítástól [6].

Ezt a folyamatot szemlélteti a 2. ábra.



2. ábra. A world switch folyamata (forrás: [20])

A tömör áttekintés után az említett processzorutasítások és -kivételek hatásait kicsit bővebben is kifejtjük.

VMRUN A VMRUN utasítás egy egyargumentumú utasítás, paraméteréül az ún. *virtual machine control block* (VMCB) fizikai memóriacímét várja. Ez a blokk a tartalmazza például az elfogandó utasítások és események listáját vagy a vendég virtuális processzorának állapotát. (A VMCB pontos felépítését megtalálja [6] B.1 fejezetében.) A VMRUN először elment bizonyos alapvető információkat a processzor állapotáról, majd betölti a virtuális gép processzorának állapotát a VMCB alapján. Amennyiben a vendég állapotának visszatöltése nem volt sikeres (inkonzisztens állapotba került a processzor), a processzor visszatér root módba. Ha sikeres volt, guest módban marad és a vendég gép kódját hajtja végre mindaddig, amíg #VMEXIT hívás nem érkezik.

Feltételezzük, hogy a VMM a globális megszakításokat a VMRUN hívás előtt letiltotta, mivel ennek a műveletnek atominak kell lennie. A visszaállítás végén a VMRUN utasítás a globális megszakításokat újra engedélyezi.

A fentieken kívül a VMRUN feladata például a virtuális gépet azonosító address space ID beállítása (erről részletesebben a 2.3. fejezetben írunk), a virtuális megszakítások maszkjának vagy a virtuális processzor timestamp counterének betöltése.

(Természetesen VMRUN utasítás csak a legmagasabb, ún. CPL-0 privilégiumszintről hívható.)

#VMEXIT Amennyiben valamilyen oknál fogva (például hiba vagy ütemezés miatt) a processzor #VMEXIT-et hajt végre, az alábbi fontosabb műveleteket végzi el a processzor: az megszakításokat globálisan letiltja, frissíti a VMCB tartalmát, visszaállítja az address space ID-t a gazdagép ID-jére (azaz nullára), visszaállítja a VMRUN által mentett processzorállapotot. Ezek után a visszatöltött adatok érvényességét ellenőrzi is.

VMSAVE, VMLOAD A VMRUN csak minimális mennyiségű állapotinformációt ment el a processzorról (pl. veremmutatókat, lapozási módot), annyit, hogy a VMM futása tudjon folytatódni a vendég kilépése után. Amennyiben ennél többre van szükség, ennek mentése megtehető a VMSAVE utasítással. A mentett állapotinformációk visszatöltésre a VMLOAD utasítás szolgál.

VMMCALL A VMMCALL utasítás lehetőséget ad arra, hogy a vendég gép explicit hívja a VMM-et. A processzor alapvetően erre vonatkozóan nem végez ellenőrzést, tehát a VMM döntheti el, hogy ez az utasítás legális vagy sem [6].

2.1.2. Az Intel VMX

Az Intel processzorok esetében a virtualizációt támogató processzorutasításokat, valamint az ezekhez tartozó vezérlő struktúrákat a VMX (*Virtual Machine Extensions*) foglalja magában. A kontextusváltások működése hasonló az AMD megoldásához, azonban kisebb szemléletbeli eltérések vannak.

A VMM szoftverek az alábbi életciklust járják be:

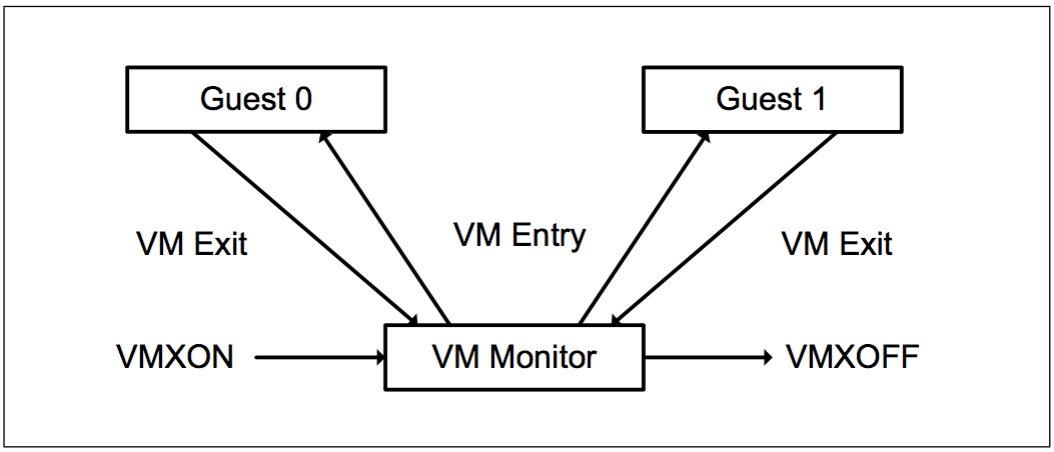
- A VMXON utasítás hatására a rendszer VMM műveletbe kezd.
- A VMM leíróiba VM Entry bejegyzéseket lehet felvenni minden virtuális géphez, melyekre a vezérlés VMLAUNCH illetve VMRESUME utasításokkal adható át.
- A VMM a vezérlést a VM Exit-ek hatására kapja vissza. Ilyenkor a VMM eltárolja, hogy miért született a VM Exit, és így eldönthető, hogy milyen utasítások végrehajtása szükséges.
- A VMM működés során többszöri VM Entry és VM Exit váltások lehetnek.
- A VMM művelet leállításához a VMXOFF utasítást kell meghívni.

Az életciklust a 3. ábra szemlélteti.

A vendég gépek vezérlése, valamint a váltás root mód és nem-root mód között a *Virtual Machine Control Structure (VMCS)* adatstruktúrák segítségével valósul meg. Ezek a struktúrák a VMCS mutató segítségével érhetőek el és a VMREAD, VMWRITE, valamint VMCLEAR utasításokkal módosíthatóak. Minden virtuális processzor esetén külön VMCS struktúra jön létre. A struktúra részletes leírása [12]-ben megtalálható.

A virtuális gépbe történő átlépés folyamatát VM Entry-nek nevezi az Intel. Ez a virtuális gép első indításakor VMLAUNCH, majd később a VMRESUME utasítások segítségével hajtható végre. Az alábbi lépésekből áll:

- Alapvető ellenőrzések a virtuális gép állapotával kapcsolatosan.



3. ábra. Az Intel VMM életciklusa (forrás: [12])

- A vezérléshez és a gazdagéphez kötődő VMCS részek ellenőrzése, a VMCS felkészítése a következő VM Exit-re.
- Processzorállapot visszatöltése és további ellenőrzések.
- Model-Specific Registers (MSR[1]) betöltése a VMCS-ből.
- Ha VMLAUNCH hívás történt, a VMCS launch state bejegyzését launched-re kell állítani.

Ez után a rendszer vezérlését a vendég gép végzi egészen addig, amíg egy VM Exit nem történik. A root módba történő visszaváltás menete:

- A kilépés okainak elmentése a VMCS megfelelő részeibe.
- A processzor állapotának elmentése a VMCS vendéggépet leíró részébe.
- MSR mentése VMCS-be.
- Gazdagép processzorának betöltése a VMCS gazda leíróiból valamint a vezérlők alapján.

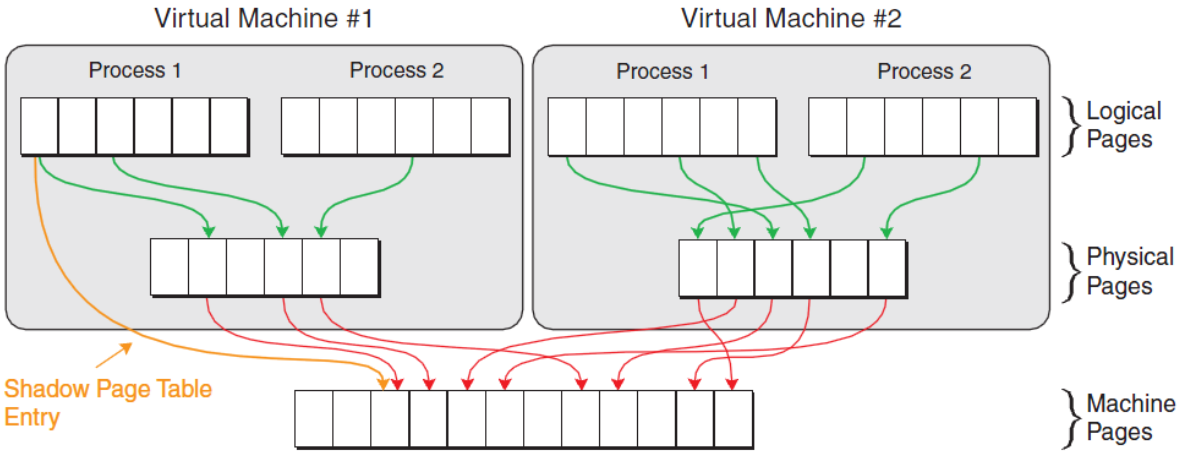
2.2. Címfeloldás támogatása

A mai, személyi számítógépeken használt operációs rendszerek esetén a futó alkalmazások a memóriából nem egy fizikai címteret látnak, hanem egy logikai címtartományt, azaz az alkalmazások virtuális címeket használnak. A virtuális címek és a fizikai címek közti összerendelés a program futása közben is változik, ezért statikus összerendelés még a futás kezdetén sem hozható létre. A virtuális és fizikai címek közti összerendelést lapszervezésű memóriakezelés esetén a laptáblák tartalmazzák. Annak érdekében, hogy a gyakran használt címeket ne kelljen minden alkalommal a laptáblákból kikeresni a processzorokban

[1] Az MSR-ek olyan vezérlő regiszterek, amelyek a konkrét processzor egyedi tulajdonságainak vezérlését teszik lehetővé

egy ún. *translation lookaside buffert* (TLB) használnak, mely egy tartalom szerint címezhető memória. Segítségével a legutóbb használt virtuális címekhez gyorsan megtalálható a megfelelő fizikai cím.

Virtuális gépek esetén kétszeres címfordítás történik: egyrészt a vendég számítógép virtuális memóriacímeit le kell képezni a vendég „fizikai" címeire (ezt a 4. ábrán a zöld nyilak jelképezik), majd pedig ezeket a gazda számítógép valós fizikai címeire (ezt a 4. ábrán a piros nyilak jelképezik). Annak érdekében, hogy a virtuális gépek esetén is ki lehessen a TLB nyújtotta támogatást használni, a szoftveres virtualizációt támogató megoldásoknál bevezették a *árnyék laptáblákat* (shadow page tables vagy *sPT*). Ennek segítségével a vendég számítógép virtuális címeit közvetlenül a gazdagép fizikai címeire lehet fordítani (ezt a 4. ábrán a narancssárga nyíl jelképezi), ezáltal a TLB is kihasználható. Azonban az árnyék laptáblákat folyamatosan szinkronban kell tartani a többi laptábla tartalmával, figyelni kell ezek módosulásait, ami pedig igen költséges, hiszen amikor a vendég gép módosítja a laptábláját, akkor ki kell lépni root módba és frissíteni kell az sPT-t is (ez bizonyos esetekben a virtualizáció által okozott teljesítményveszteség 75 %-át teszi ki [3]). Problémás az is, hogy ez a plusz laptábla igen komoly méretű is lehet.



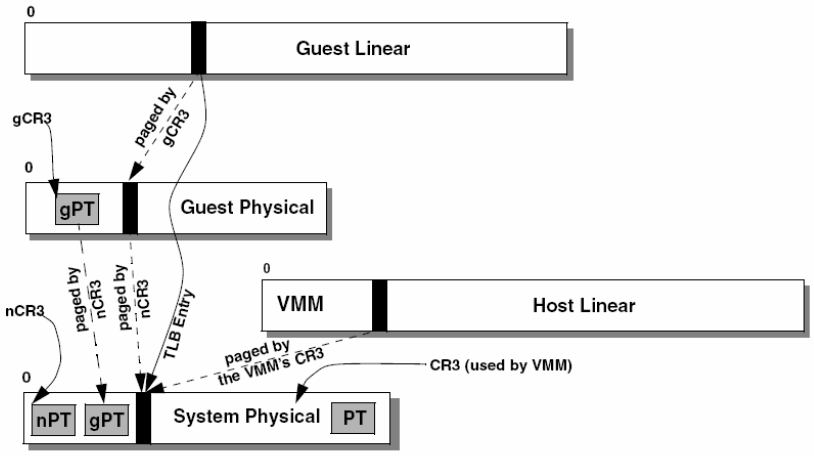
4. ábra. Címfordítás árnyék laptáblák használatával (forrás: [19])

Ezt az overheadet igyekszik csökkenteni az *AMD RVI* (Rapid Virtualization Indexing), más néven *NPT* (Nested Paging Tables), illetve az *Intel EPT* (Extended Page Table) technológiája. Ezek segítségével a vendég gép virtuális címeinek a gazdagép fizikai címeire leképezése hardveres támogatást kap anélkül, hogy árnyék laptáblákat kellene használni [13].

Mindkét technológia használata esetén két aktív laptábla van: egyik a vendég virtuális címterét képezi le a vendég fizikai címterére (ezeket nevezik *Guest Page Tables*-nek vagy *gPT*-nek), másik pedig a vendég fizikai címterét a gazdagép fizikai címterére képzi (ezt nevezi az AMD *Nested Page Tables*-nek vagy *nPT*-nek, míg az Intel *Extended Page Table*-nek vagy *EPT*-nek).

A processzor feladata egy vendég gépből történő címleképzés esetén, hogy bejárja a *gPT*-t majd az *nPT*/*EPT*-t, tehát a gazdagép tábláit kihagyhatja, így gyorsabb lehet a leképzés, valamint hatékonyabb lehet a TLB-k használata. Az *sPT*-vel szemben azért hatékonyabb ez a megoldás [15], mert amennyiben egy vendég gép frissíteni szeretné a laptábláját, akkor ezt nyugodtan megteheti, anélkül, hogy a VMM-nek működésbe lépne, azaz root módba kellene váltani.

Az RVI működését mutatja be az 5. ábra.



5. ábra. AMD RVI működése (forrás: [20])

A technológiák használata a vendég operációs rendszerek számára teljesen transzparens.

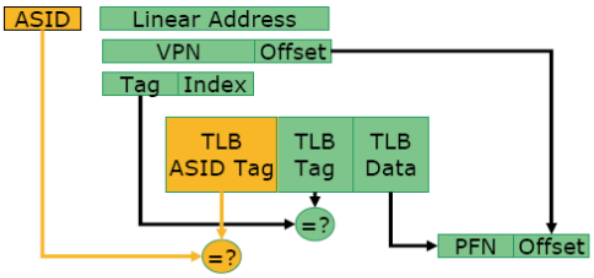
A VMware mérései alapján az RVI használatával memóriaintenzív benchmarkok esetén akár ötszörös, valós alkalmazások esetén akár 20–30 %-os gyorsulás érhető el [19].

2.3. TLB-t érintő technológiák

A translation lookaside buffer (TLB) egy olyan processzorbeli cache, mely virtuális címek fizikai címmé fordítását gyorsítják. Az ötlet a TLB mögött az, hogy egy futó program gyakran ugyanazt a memóriaterületet használja újra és újra. Ezért elég ezt egyszer kikeresni, majd utána a TLB-ben eltárolni. Amennyiben a TLB egy virtuális cím fizikai párját tartalmazza, úgy szükségtelenné válik a virtuális cím nagy méretű laptáblából történő költséges kikeresése. Mivel a TLB tartalom szerint címezhető, a benne történő keresés nagyon gyors. A mai processzorok esetén a TLB-k találati aránya tipikusan igen magas, 80–90 % körüli.

Ha azonban egy virtuális gép futását megszakítja egy másik virtuális gép, akkor a TLB-ben szereplő értékek érvénytelenné válnak, így azt üríteni kell [20]. Tehát minden alkalommal, amikor futási jogot kap egy virtuális gép, üres TLB-ből kell kiindulnia. Ez jelentős teljesítménycsökkenést okozhat. Ennek elkerülésére szolgálnak a *VPID* (Intel) és *Tagged TLB* (AMD) technológiák. Mindkettő lényege az, hogy a TLB-ben szereplő bejegyzéseket ellátják egy cimkével is, amely segítségével eldönthető, hogy az adott bejegyzés melyik virtuális géphez (vagy a hoszt géphez) tartozik, így a bejegyzések nem keverednek össze. Ezáltal nem szükséges a TLB tartalmának törlése virtuális gépek váltása esetén, így maximalizálva a virtuális gép által használható TLB-bejegyzések számát, amikor egy virtuális gép kerül futó állapotba [13]. Így csökken a virtuális gépek váltása által okozott overhead [5].

Az AMD Tagged TLB megoldását mutatja a 6. ábra. A TLB bejegyzései kiegészültek egy Address Space ID (ASID) címkével. Minden TLB-keresés során ez az ASID címke is összehasonlításra kerül az aktuálisan futó virtuális gép ASID-jével. Így az egyes virtuális gépekhez tartozó bejegyzések egymástól megkülönböztethetők.



6. ábra. Tagged TLB (forrás: [3])

Nyilvánvaló, hogy a tagged TLB technológia akkor használható ki, ha megfelelően sok bejegyzés tárolható a TLB-ben. Annak érdekében, hogy minél több hasznos bejegyzés lehessen a TLB-ben [13] szerint az AMD kiterjesztett TLB-t használ az AMD-V keretében, amellyel az L1 cache-ben 48, az L2 cache-ben 128–512 bejegyzés tárolható Opteron processzorok esetén (2009-es adat). Megjegyzendő azonban, hogy már a K7 sorozatú, asztali számítógépekbe szánt processzorok is hasonló méretű TLB-vel rendelkeztek évekkel az AMD-V technológia bejelentése előtt (L1 cache-ben 24–40 adatbejegyzés, L2-ben 256 adatbejegyzés).

Az összehasonlíthatóság kedvéért érdemes megnézni, hogy az Intel mekkora TLB-ket illeszt a mai processzoraiba. Az aktuálisan legfrissebbnek mondható Nehalem magos Intel Core i7 processzorok kétszintű TLB-vel rendelkeznek [10]. Az első szinten külön van adat és utasítást TLB. Az adattár 64 kisméretű vagy 32 nagyméretű lapra tud hivatkozni. A processzor első szintű TLB-jében 128 kisméretű utasítás lap tárolható (nagyméretűből mindössze 7). A második szinten már egyesítve van az adat és az utasítás tár és kizárólag kisméretű lapokkal működik, amelyből 512-t tud tárolni.

2.4. Migráció támogatása

A virtualizáció elterjedésével és egyre nagyobb mértékű felhasználásával megjelent az igény arra, hogy a futó virtuális gépek rövid idő alatt, szinte észrevétlenül átmozgathatók legyenek egyik fizikai gépről egy másikra. A virtuális gépek leállítása és újraindítása nem megfelelő megoldás, mivel jelentős szolgáltatáskiesést okoz. Az élő migrálás (*live migration*) során a fizikai hosztok közti váltás során a virtuális gépek nem kerülnek leállításra. Így viszont problémát jelent, ha a migráció előtti és a migráció utáni környezet eltér egymástól – ez fokozottan igaz a processzorokra. Az élő migráció az alkalmazások számára nem látható, viszont előfordulhat, hogy a migráció utáni fizikai processzor kevesebb képességgel rendelkezik a korábbi fizikai hosztnál, ez váratlan hibákat okozhat a program futásában [4].

Ennek megoldására az Intel és az AMD is kidolgozott egy-egy technológiát: az Intel a *FlexMigration*-t, az AMD pedig az *AMD-V Extended Live Migration Technology*-t. Mindkettő lényege, hogy a virtuális gép elől elrejti a processzor valós képességeit, ehelyett a szerverfarm összes processzorának tudásából csak ezek metszetét mutatja a virtuális gépek számára. Ezt a CPUID módosításának segítségével érik el. A CPUID a processzor képességeinek, funkcióinak leírására szolgál, a szoftvereknek ez alapján kell megállapíta-

ni, hogy milyen utasításokat használhatnak.[2] A VMM-ek minden virtuális gép számára külön-külön meghatározhatják a CPUID értékét.

Az AMD 2003-ban vezette be AMD-V Extended Live Migration Technology-t az Opteron processzoraiban. Ezekben már rendelkezésre állnak CPUID Override Model Specific Registerek, melyekkel meghatározható a virtuális gépek felé mutatni kívánt CPUID. Érdekessége az AMD megoldásnak, hogy a CPUID változtatásához először egy meghatározott regiszterbe egy jelszót kell másolni. A Model Specific Registerek módosítására külön processzorutasítások is vannak (RDMSR, WRMSR).

Az Intel 2007-ben mutatta be az első élő migrációt támogató processzorát, amely működésében csak nagyon apróságokban tér el az AMD technológiájától.

[9]-ben megmutatták, hogy bizonyos esetekben lehetőség van élő migrációra akár Intel és AMD processzorok között is. Ez azonban bizonyos utasítások emulációját kívánja meg.

Fontos, hogy ezek a technológiák sem oldanak meg minden migrációs problémát. Még a virtuális gépek leállított állapotú átmozgatása (azaz a cold migration) is hibalehetőségeket rejt magában, mivel az operációs rendszer egyes részei erősen hardverspecifikusak lehetnek és nem képesek már alkalmazkodni a megváltozott hardverkörnyezethez telepítés után.

2.5. Biztonsági technológiák

A virtualizációs technológiák ma már jelentős teret hódítottak maguknak a szerverek, valamint a missziókritikus rendszerek világában is. Olyan területeken is gyakorta alkalmazzák ezeket a technológiákat, ahol a rendszer kiemelkedő védelmet igényel a támadások, valamint a szoftverhibák ellen is. Mindkét nagy processzorgyártó cég kifejlesztett erre a célra néhány technológiát, amelyeket ebben a fejezetben szeretnénk bemutatni.

Az Intel elsődleges biztonsági technológiája a *Trusted Execution Technology* (TXT), amelynek célja, hogy egy biztonságos futási környezet legyen létrehozható a rendszerben. Egy rendszerben akár több ilyen biztonságos környezet is definiálható, amelyekhez hozzárendelhetők a használt dedikált erőforrások és amelyeket a processzor, a chipset és az operációs rendszer vagy a VMM kezelhet.

Az Intel TXT technológia lehetővé teszi, hogy a rendszer indulási folyamatát is ellenőrzés alá vonjuk. Lehetségessé válik az is, hogy a ellenőrizzük, a VMM a megfelelő alaphelyzetbe került-e indulás után.

Az Intel cég egy másik fontos védelmi vonala a *Descriptor Table Exiting* technológia. A Descriptor Table (DT) a memóriaszegmensek leírását tartalmazza. Ezek a vendég rendszeréből védettek, de azon kívülről nem, így ehhez külön védelmet kell megvalósítani. A technológia segítségével a memóriszegmensek leíróinál beállítható, hogy módosítások esetén VMExit hívás történjen, tehát a VMM értesüljön az esetleges támadási kísérletekről.

Az AMD megoldásának részét képezi a *Device Exclusion Vector*ok (DEV) használata. Ezek segítségével minden eszköz DMA művelete előtt eldönthető, hogy az adott eszköznek van-e jogosultsága az adott művelet elvégzésére. A DEV minden 4 KB-os fizikai laphoz 1 biten eltárolja, hogy az adott eszköz hozzáférhet-e a laphoz vagy sem. Egy rendszerben 4 Protection Domain létezik, ezek mindegyikéhez tartozik egy-egy DEV [20][1].

Később jelent meg az AMD átfogóbb biztonsági technológiája, az *AMD Presidio*. Ez magában foglalja a DEV használatát, de emellett többek közt tartalmazza a modellspe-

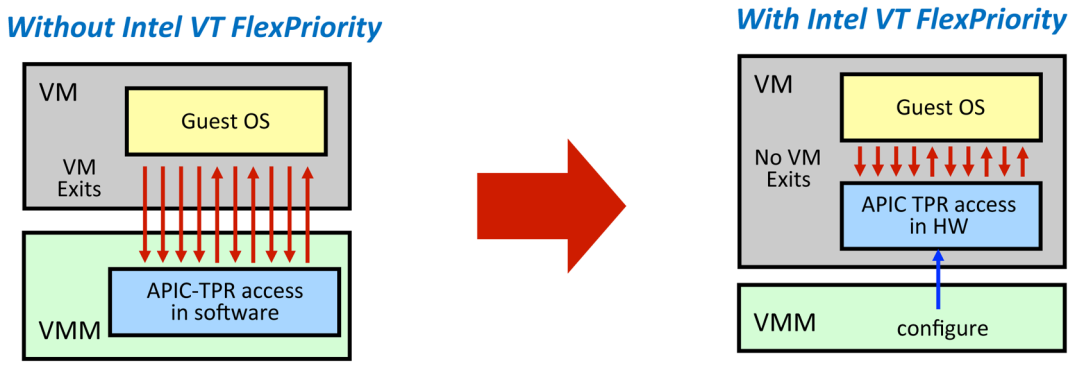
[2]Megjegyzendő azonban, hogy más módszerekkel is kideríthetők a processzor képességei. Ezek a módszerek az AMD által nem támogatottak és az AMD-V Extended Live Migration Technology nem is nyújt rá megoldást. Így ez a megoldás a Popek és Goldberg által leírt követelményeket nem teljesíti.

cifikus regiszterek védelmét a virtuális gépekkel szemben egy MSR Protection Map alkalmazásával, valamint tartalmaz egy az Intel TXT-hez hasonló technológiát [17]. Az AMD Presidio technológiáról kevés leírás érhető el, részletes ismertetése gyakorlatilag csak a processzorok manualjában található meg [6].

2.6. FlexPriority

Az Intel FlexPriority technológiája a virtualizált környezetben történő megszakítás kezelést hivatott támogatni [11]. A mai processzorokban a megszakításokat a processzoronkénti *APIC*, azaz az *Advanced Programmable Interrupt Controller* kezeli. A vezérlőben természetesen rengeteg féle regiszter található, azonban amellyel a FlexPriority technológia foglalkozik az a *Task-Priority Register* (TPR). A regiszter feladata [8], hogy számon tartsa az éppen aktuálisan futó feladat prioritását. A kernel minden egyes kontextusváltásnál frissíti a regiszter tartalmát. Amennyiben egy megszakítás érkezett egy vendég operációs rendszerhez, annak ki kell olvasnia a TPR tartalmát. Ez a FlexPriority technológia nélkül egy VM Exit-et igényelne, azaz ki kéne lépni root módba, ami igen költséges. A FlexPriority azonban lehetővé teszi, hogy a regiszter tartalmát közvetlenül a vendég is elérje (a 7. ábra). Ehhez a memóriába egy virtuális TPR van leképezve, amely olvasásához soha nem kell root módba lépni és írás esetén is csak bizonyos esetekben szükséges ez. Mivel egyes operációs rendszerek (mint például a Windows XP) igen gyakran olvassa és írja a TPR-t, így ez nagyon komoly gyorsulást jelenthet. Az Intel saját mérései szerint egy Windows XP operációs rendszerű vendég gép 40%-kal gyorsabban indul el a technológiának köszönhetően [11].

Tudomásunk szerint az AMD-nek külön megnevezett, ehhez hasonló funkciót megvalósító technológiája nincs.

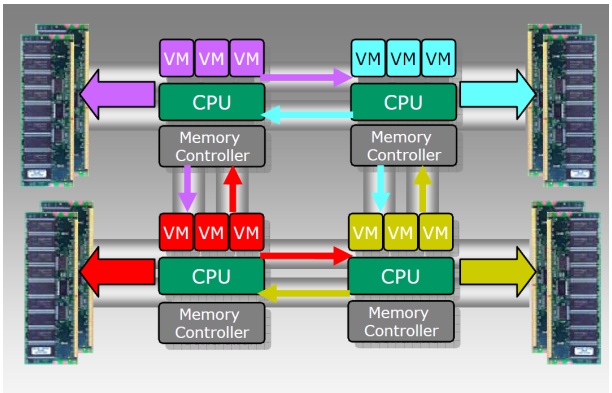


7. ábra. A TPR regiszter elérése FlexPriority technológia nélkül és FlexPriority-vel (forrás: [15])

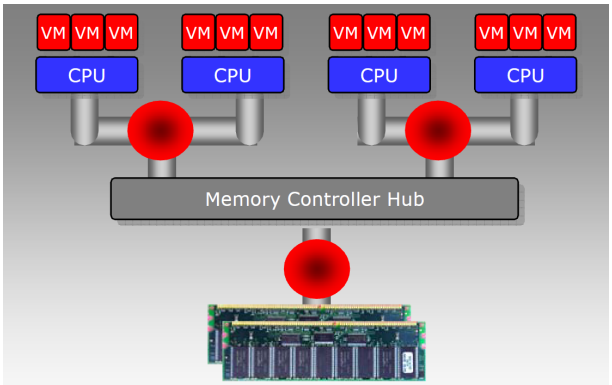
2.7. Egyéb technológiák

Az AMD számos technológiáját sorolja a virtualizációt támogató képességek közé. Ilyen például a *Direct Connect Architecture* (DCA), melynek segítségével skálázható rendszerek építhetők, amelyek nagy terhelést is képesek kiszolgálni. A front side bus architektúra helyett a processzor, memória és az I/O eszközök közvetlen összeköttetésben állnak, mely

kis késleltetést és hatékony memóriaelérést eredményez [5]. A DCA elemei a következők: 64 bites memóriacímzés, többmagos processzorok, HyperTransport technológia, processzorokba integrált memóriavezérlő. Ezek közül talán a legutolsó a legfontosabb. Mivel az AMD processzorokba integrálva van a memóriavezérlő, a memóriakérések nem a memóriabuszon keresztül jutnak el a memóriavezérlőhöz. Ráadásul a NUMA (non-uniform memory architecture) architektúra miatt a memóriakérések kiszolgálása elosztott: minden egyes processzorhoz saját memóriaterület tartozik és a memóriavezérlőjük csak ezzel a területtel foglalkozik [13]. A Direct Connect Architecture és a megosztott FSB-alapú architektúra közti különbségek láthatók a 8. ábrán.



(a) Direct Connect Architecture

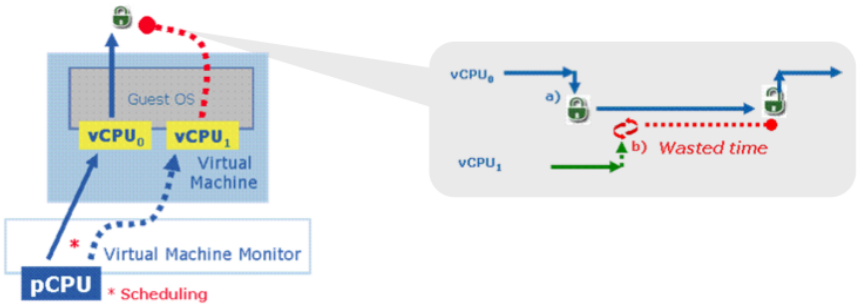


(b) Megosztott front side bus alapú architektúra

8. ábra. A Direct Connect Architecture és a megosztott FSB-alapú architektúra közti különbségek (forrás: [20])

Az Intel több különböző egyedi virtualizációs technológiát is kidolgozott, amelyek segítségével a virtualizált környezetben működő rendszereket még gyorsabbá és stabilabbá tehetőek. Az első, amelyet ebből kiemelnénk a *Guest Preemption Timer* technológia. Ennek célja, hogy a VMM-et készítő cégek kezébe egy olyan eszközt adjon, amellyel jobb minőségű, illetve flexibilisebb szolgáltatást nyújthatnak, valamint QoS garanciákat vállalhatnak. A technológia működése rendkívül egyszerű: VM Enter előtt a VMM egy visszaszámlálót állít be. Amikor az óra lejár, egy VMExit történik a megszakításkezelő rendszerektől teljesen függetlenül. Enek segítségével garantálható, hogy a VMM időnként bizonyos utasításokat lefuttathasson függetlenül a vendég rendszer típusától és működésétől.

Az Intel egy másik technológiája a *Pause-Loop Exiting*, amely többszálú ütemezés problémáit hivatott orvosolni [7]. A 9. ábrán látható az alapvető probléma, amely megoldására a technológiát megalkották. Amennyiben egy program egy szála egy erőforrást lefoglal, lockot hoz létre rá, majd ezt az ütemező preemptálja és egy olyan másik szálnak adja a futást, amely szintén azt az erőforrást használná, akkor mindkét szál várakozásra kényszerül. A probléma ezzel az, hogy a második szál a processzort használja, annak ellenére, hogy tovább futni nem képes (*spinlock* jön létre), ez okozza a 9. ábrán is megjelölt elpocsékolt processzoridő.



9. ábra. Erőforrásfoglalás közbeni preempció (forrás: [7])

A probléma megoldásához szükséges, hogy ezeket a felesleges várakozásokat a VMM-ből érzékelhessük. Ehhez szükséges azt tudnunk, hogy a spinlock-ok megvalósításánál a processzor Pause hívásokat kap egymás után. A Pause-Loop Exiting lehetővé teszi, hogy a VMM-ben meghatározhassunk, hogy egymás után mennyi időnként várunk Pause hívásokat (*gap*), valamint azt is, hogy ilyen Pause-ciklusokban mennyi időt tölthetünk VM Exit nélkül (*window*). Amennyiben a meghatározott értékeket eléri a rendszer, akkor a VMM egy VMExit hívást küld, így lehetővé teszi, hogy a VMM újraütemezhesse a szálakat (tehát az erőforrást foglaló szálat engedje futni). Az Intel mérései szerint a technológia jelentős gyorsulást eredményez olyan környezetekben, amikor a rendszerek erősen terheltek [7].

3. Összefoglalás

A dolgozatban röviden bemutattuk az Intel és AMD által processzoraikban alkalmazott főbb technológiákat, melyekkel a virtualizációt hardveresen támogatatják. Ahogyan a processzoraik egyéb tulajdonságaiban, úgy a virtualizáció támogatása terén is fej-fej mellett halad a két gyártó. Az alkalmazott technológiáik nagyon hasonlóak, nincsenek igazán jelentős különbségek és nem lehetne egyértelműen győztest hirdetni közöttük.

A dolgozat írása közben azzal is szembesültünk, hogy a konkrét virtualizációs technológiákat a processzorgyártók nem verik nagy dobra. Természetesen a marketinganyagokban megjelenik a zászlójukra tűzve a virtualizáció szó, azonban kevés olyan anyag érhető el, amely a konkrét megoldásokat részletezné. A processzorok fejlesztői leírásaiban ([6] és [12]) pontosan leírásra került az egyes technológiák alkalmazása, azonban egyetlen informatikai döntéshozótól sem várható el, hogy ezeket a többezer oldalas dokumentációkat áttanulmányozzák.

Ábrák jegyzéke

1. Hardveres virtualizáció esetén létező CPU Ring-ek (forrás: [18]) 4
2. A world switch folyamata (forrás: [20]) 5
3. Az Intel VMM életciklusa (forrás: [12]) 7
4. Címfordítás árnyék laptáblák használatával (forrás: [19]) 8
5. AMD RVI működése (forrás: [20]) 9
6. Tagged TLB (forrás: [3]) 10
7. A TPR regiszter elérése FlexPriority technológia nélkül és FlexPriority-vel (forrás: [15]) 12
8. A Direct Connect Architecture és a megosztott FSB-alapú architektúra közti különbségek (forrás: [20]) 13
9. Erőforrásfoglalás közbeni preempció (forrás: [7]) 14

Hivatkozások

[1] Advanced Micro Devices, Inc. AMD "Pacifica" Virtualization Technology. 2005. http://www.xen.org/files/Xen_PacificaDisclosure_AMD_EWahlig.pdf.

[2] Advanced Micro Devices, Inc. AMD64 Virtualization Codenamed "Pacifica" Technology – Secure Virtual Machine Architecture Reference Manual. 2005. http://www.mimuw.edu.pl/~vincent/lecture6/sources/amd-pacifica-specification.pdf.

[3] Advanced Micro Devices, Inc. AMD-V^{TM}Nested Paging. 2008. http://developer.amd.com/assets/NPT-WP-1%201-final-TM.pdf.

[4] Advanced Micro Devices, Inc. Live Migration with AMD-V^{TM}Extended Migration Technology. 2008. http://developer.amd.com/assets/43781-3.00-PUB_Live-Virtual-Machine-Migration-on-AMD-processors.pdf.

[5] Advanced Micro Devices, Inc. Virtualizing server workloads. *AMD White Papers*, 2008. http://www.amd.com/us/Documents/AMD_WP_Virtualizing_Server_Workloads-PID.pdf.

[6] Advanced Micro Devices, Inc. *AMD64 Architecture Programmer's Manual, Volume 2: System Programming*. 2010. http://support.amd.com/us/Processor_TechDocs/24593.pdf.

[7] Bing Wang. From Virtualization 2.0 to Enterprise Cloud: Usages and Technologies. 2010. http://software.intel.com/file/27289.

[8] Daniel P. Bovet and Marco Cesati. Understanding Linux Kernel 3rd edition. 2006. http://book.opensourceproject.org.cn/kernel/kernel3rd/opensource/0596005652/main.html.

[9] Uwe Dannowski and Andre Przywara. Cross-vendor migration. 2010. http://developer.amd.com/assets/CrossVendorMigration.pdf.

[10] Fedy Abi-Chahla (Tom's Hardware). Intel Core i7 (Nehalem): Architecture By AMD? 2008. http://www.tomshardware.com/reviews/Intel-i7-nehalem-cpu,2041.html.

[11] Gideon Gerzon. Intel® Virtualization Technology Processor Virtualization Extensions and Intel® Trusted execution Technology. 2007.
http://software.intel.com/file/1024.

[12] Intel, Inc. *Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3B: System Programming Guide, Part 2*. 2010.
http://www.intel.com/products/processor/manuals/.

[13] Tim Mueting. Why Virtualization Runs Faster on AMD Opteron™Processors. 2009.
http://developer.amd.com/documentation/articles/pages/whyvirtualizationrunsfasteronamdopteron.aspx.

[14] Gerald J. Popek and Robert P. Goldberg. Formal requirements for virtualizable third generation architectures. *Commun. ACM*, 17:412–421, July 1974.

[15] Rich Urlig. A Primer on Virtualization. 2009.
http://software.intel.com/file/26677/.

[16] Marco Righini. Enabling Intel® Virtualization Technology – Features and Benefits. 2010.
http://www1.euro.dell.com/content/topics/global.aspx/power/en/intel_virtual?c=be&l=fr&s=gen.

[17] Geoffrey Strongin. Trusted computing using AMD "Pacifica" and "Presidio" secure virtual machine technology. 2005.
http://os.korea.ac.kr/course_papers/2009_spring/TCB-amd.pdf.

[18] Tóth Dániel, Micskei Zoltán. Virtualizációs Technológiák és Alkalmazásai, CPU és memória virtualizáció. 2010.
https://www.inf.mit.bme.hu/content/cpu-%C3%A9s-mem%C3%B3ria-virtualiz%C3%A1ci%C3%B3.

[19] VMware, Inc. Performance Evaluation of AMD RVI Hardware Assist. 2009.
http://www.vmware.com/pdf/RVI_performance.pdf.

[20] Elsie Wahlig. AMD virtualization technology. 2006.
http://streaming.linux-magazin.de/virtualization/ewahlig/pdf/ewahlig.pdf.