RealVM

A PROJEKTRŐL

Ez a projekt egy új típusú virtuális gép kidolgozását taglalja, amely képes több operációs rendszer egyidejü futtatására, valamint az operációs rendszerek közötti gyors váltásra. A valódi központi vezérlőegységet (processzort) használja az emulált gép szoftverének futtatására, és képes a számítógép valódi hardverének haszálatára. Az emulált operációs rendszer magját (kernelét, a továbbiakban emulált kernel) felhasználói módban futtatja. Ez lehetővé teszi számára az emulált kernel működésének nyomon követését, ami által lehetővé válik több operációs rendszer futtatása egy gépen, valamint az operációs rendszerek közötti gyors váltás. Az ilyen típusú megoldás gyosabb az eddigieknél, mivel csak a számítógép harverének bizonyos, nélkülözhetetlen részeit emulálja (a memóriakezelés, a DMA, a PCI bizonyos részeit, stb). Nagy előnye a mai virtuális gépekkel szemben továbbá, hogy a számítógép valódi hardverét tudja használni a virtuális gépekben. Ez nagy segítséget jelenthet olyan porgramozók számára, akik több operációs rendszeren működő programokat fejlesztenek. Azok számára is sokat segíthet, akik például Linux operációs rendszert használnak, viszont nincsenek meghajtóprogramjaik bizonyos harverelemekhez. Meghajtóprogramok tesztelésénél és hibakeresésnél is kiválóan alkalmazható.

I. BEVEZETŐ

A kernelekről

A kernel az operációs rendszer magja. Kezeli a számítógép hardverét, a memóriát, a szálakat, bizonyos szolgáltatásokat nyújt a programok részére (pl.: írás a merevlemezre, stb). A kernel úgynevezett kernel módban fut. Ez azt jelenti, hogy teljes és kizárólagos hozzáférése van a számítógép hardveréhez. A felhasználói programok felhasználói módban futnak. Ha egy felhasználói módban futó program a számítógép valamely hardveréhez próbál hozzáférni, a kernel általában blokkolja azt, és kilép a programból. Maga a processzor tartalmazza beépitve azt a védelmi mechanizmust, amely megakadályozza a hardverhozzáférést a felhasználói programok részére. Ez azért szükséges, mivel ellenkező esetben több program férhetne hozzá egyidőben ugyanazon hardverhez, veszélyeztetve a rendszer stabilitását és biztonságát. Az felhasználói mód által a kernel nyomonkövetheti a felhasználói programok futását a kivételkezelés (exception handling) segítségével.

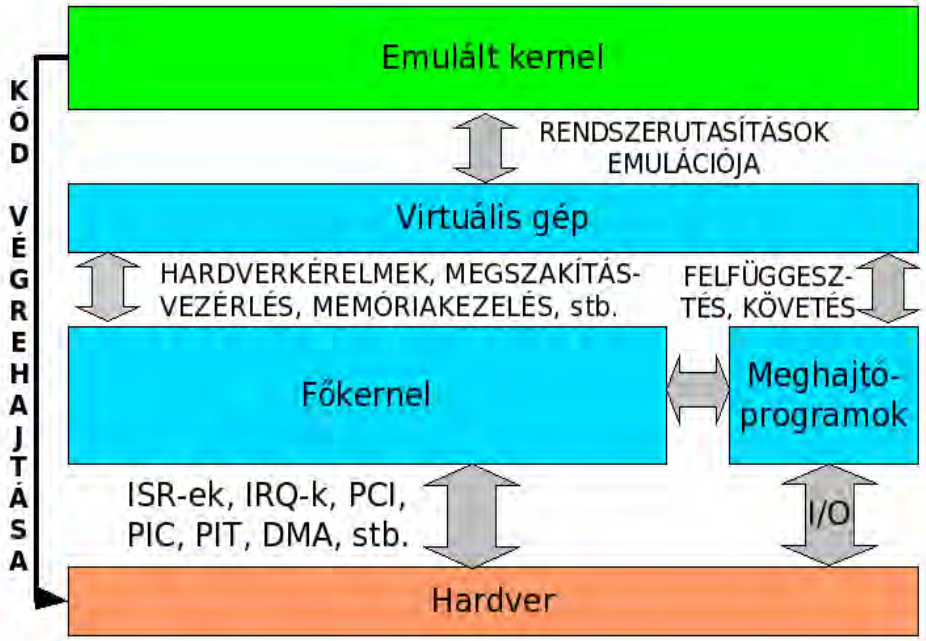
Kernel felhasználói módban

A projektum alapötlete a kernel felhasználói módban való futtatása. Ezzel lehetővé válik az alkernel futásának nyomonkövetése, ami pedig lehetővé teszi, hogy az alkernelt a számítógép valódi processzorán futtassuk. Lehetőség nyílik a futó kernelek programokként való kezelésére, azaz képesek vagyunk azokat suspendelni, schedulolni, stb. A számítógép valódi hardvere feloszthatóvá és használhatóvá válik a futó virtuális gépek között.

II. KIDOLGOZÁS

A virtuális gép egy, erre a célra kifejlesztett kernelen fut. A számítógép induláskor futtatja ezt a kernelt (a továbbiakba főkernel), amely azután beolvassa a telepített operációs rendszer magját (emulált kernelt), és azt felhasználói programként futtatja. Ezáltal megtartja magának a hardverhozzáférés és memóriakezelés jogát, így az emulált kernel számára észrevehetetlen maradhat a memóriában.

A virtuális gép felépítése az 1. ábrán látható.



1. Ábra: A virtuális gép felépítése

A virtuális gép úgy lett tervezve, hogy képes legyen az emulált operációs rendszer magjának a valódi processzoron való futtatására. Ez azt jelenti, hogy nem a teljes processzort emulálja, mint egyes virtuális gépek. A mai virtuális gépek közül is sok ezt teszi, viszont mivel ezek mind felhasználói módban futnak, egy általános célú operációs rendszer alatt, ezért ezeknek a parancsok (instrukciók) sokkal nagyobb hányadát kell emulálniuk, sokkal összetettebb módon. Ez nagymértékben kihat a sebességükre . Amikor az emulált kernel valamely harverelemhez próbál hozzáférni, a processzor ezt érzékeli (mivel az emulált kernel felhasználói processz), és kivételt hoz létre. Ezt a kivételt használja fel a virtuális gép a szükséges utasítások emulálására vagy pedig bizonyos memóriakezelési műveletekre. Az ilyen kivételek által lehetővé válik a számítógép hardverállapotának nyomonkövetése is, amely lehetővé teszi a meghajtóprogramok számára a hardverelemek állapotának igen gyors mentését. Ezáltal lehetővé válik az operációs rendszerek közötti gyors váltás, ami az operációs rendszerek schedulelését is lehetővé teszi (a hardverállapot mentése csak a pillanatnyilag futó kernel cseréjekor szükséges, ha a hardverelemhez több alkernel is hozzáférhet. Ha a hardverelem úgy van elosztva a virtuális gépek között, hogy az új virtuáis gép nem férhet hozzá, ez nem szükséges). Ennek a megoldásnak az a hátránya, hogy minden hardverelemhez meghajtóprogramot igényel.

Felmerülő problémák

A legnagyobb nehézség e projektumban annak a megvalósítása, hogy a kernel a memóriában tudjon maradni, mégpedig az emulált kernel által teljesen észrevétlenül. Ez okból kifolyólag a processzor memóriakezelőjének bizonyos részeit muszály emulálni. Az egész projektumban ez a rész hat ki legjobban a virtuális gép futásának sebességére. A kernel elrejtésére több módszer létezik, attól függően, hogy milyen az emulált virtuális gép 16, 32 vagy 64 bites módban fut. Ha 32 bites módban fut, akkor ez úgy oldható meg, hogy a fő kernelt a virtuális címtartomány 4. gigabájtja fölé kell linkelni. Az emulált kernel ring 1 vagy 2-ben fut. Mivel ehhez a címtartományhoz 32 bites címzéssel már nem lehet hozzáférni, ezért a fő kernelnek nem kell semmilyen külön mechanizmust alkalmaznia. 64 bites módban viszont az ilyen megközelítés problémákba ütközik, mivel az emulált kernel hozzáférhet a teljes virtuális címtérhez. Így az emulált kernelt ring 3-ban kell futtatni, a főkernel által használt memóriaterületeket pedig supervisor lapoknak kell jelölni. Memóriahozzáféréskor ezeken a

területeken kivételek jönnek létre és az emulátor emulálhatja a szükséges instrukciókat, gondot fordítva a virtuális címek más címekre való átirányítására. A ring 3-ban való futásból következik, hogy a teljes védett mód nem fog működni, mivel a virtuális gépen futó bármely program hozzáférhet a virtuális gép címtartományának minden részéhez. Ez legtöbb esetben nem jelent gondot, mivel az otthoni felhasználóknak, valamint a tesztelőknek és programozóknak akik a virtuális gépeket használják nincs szükségük ezekre a védelmi mechanizmusokra. Ha viszont mindenáron szükséges a teljes védett mód, akkor ez az emulált operációs rendszer kernelének teljes emulációjával érhető el, ami viszont a virtuális gép sebességét nagymértékben csökkenti. A védett mód többi részét ez a módosítás nem befojásolja, hiszen az összes többi védelmi mechanizmus továbbra is megmarad, vagy a valódi processzoron, vagy a kivételek létrejöttekor az emulátoron végbemenő ellenőrzés eredményeként. Külön figyelmet kell szentelni a fő kernel báziscímének megválasztásakor, hogy a kernel olyan címtartományba kerüljön, melyet a legtöbb operációs rendszer ritkán használ (azaz a kernel és a programok memóriatartományán kívül). Ezzel elérhetjük, hogy a lehető legkevesebb instrukció emulálására legyen szükség.

Azon harverelemek emulációja is szükséges, amelyeket kizárólagosan hozzárendeltünk valamely virtuális géphez, viszont minden virtuális gép működéséhez nélkülözhetetlen (ilyen lehet például a videóvezérlő).

A pillanatnyilag futó virtuális gép megváltóztatásához szükséges a hardverelemek állapotának nagyon gyorsan történő mentése és visszállítása. Ehhez szükség van arra, hogy a fő kernel és a meghajtóprogramok nyomonkövessék a hardverelemek állapotát. Ezt viszont olyan módon kell megvalósítaniuk, hogy minnél kevésbé lassítsák a virtuális gép futását . Ezért követni kell az adatátvitelt bizonyos portokon és címterületeken.

Megvalósítás

A kernel és a virtuális gép C és assembly nyelven lett írva. x86_64 architektúrára készült (AMD64 és IA64). Két fő részből tevődik össze:

- A főkernel, amely futtatja a virtuális gépet
- Virtuális gép, amely egy processz a főkernelen. Feladata az emulált kernelek kéréseinek kiszolgálása és a nélkülözhetetlen instrukciók emulálása.

A főkernel

A főkernel e projekthez lett kifejlesztve. A főkernel tartalmazza a szükséges alapokat a számítógép hardverének és a processzek kezeléséhez (memory kezelés, ISR, IRQ kezelés, scheduler, meghajtóprogramok a merevlemez kezeléséhez, a PIChez, a PIThez, stb). Úgy lett tervezve, hogy könnyen együtt tudjon működni a virtuális géppel.

A memóriakezelő úgy lett tervezve, hogy a virtuális gép számára lehetővé tegye a laptáblázatokhoz való könnyű és gyors hozzáférést. Némely műveletek sebességére, mint például a laptáblázatok törlésére, külön figyelment kell szentelni. Mivel ezek a műveletek igen sűrűn jelentkeznek, ezért nagyon gyorsan kell működniük.

Az ISR és IRQ kezelő úgy lett tervezve, hogy bármely megszakításhoz hozzátudjon rendelni bármilyen kezelőt, akár a virtuális gép operációs rendszeréből, akár magából a főkernelből.

A driverek PIO módban működnek. Ez nagy előnyt jelent, mivel így a DMA csatornák a virtuális gépek rendelkezésére állnak, valamint az alsó 16 megabájt memória is szabadon marad.

A számítógép hardverének detektálása a PCI busz szkennelésével történik. A hardverelemekről eltárolódnak bizonyos adatok, melyek alapján a virtuális gép képes emulálni a PCI buszt szkenneléskor és a PCI BIOS-t. Ez lehetővé teszi bizonyos hardverelemek elrejtését a virtuális gép operációs rendszere elől, lehetővé téve a számítógép hardverének elosztását a virtuális gépek között.

A rendszerindítás

A rendszerindítás az operációs rendszer betöltését jelenti. Amikor a bootloader beolvassa a főkernelt, az elindít egy virtuális gépet, majd a merevlemezről, vagy annak egy partíciójáról beolvassa a telepített rendszer bootloaderét és elindítja a virtuális gépet. A bootloadert 16 bites módban kell indítani, ami további nehézségekhez vezet, mivel a főkernel 64 bites módban fut. 64 bites módban nem lehet 16 bites programokat futtatni, mivel nem támogatja a virtual 8086 módot. Ezért szükséges a 16 bites mód teljes emulációja. Amikor a virtuális gép detektálja, hogy a bootloader aktiválja a 32 bites módot, elkészíti a szükséges leírótáblákat a valódi processzoron és a virtuális gép szoftverének futtatását átadja a valódi processzornak. Ezt a 32bites compatibility mód segítségével éri el. Ha az operációs rendszer átlép 64 bites módba, a virtuális gépnek csak a szegmensek módosítására van szüksége és folytathatja a virtuális gép szoftverének futtatását.

A 16 bites mód emulációja

A 16 bites mód emulációja úgy történik, hogy a virtuális gép egymás után olvassa az utasításokat a kódszegmensből, és emulálja azokat. Az emulációt ugyanaz az emulátor végzi, amey 32 és 64 bites módban is használatos, csupán néhány paraméterben különbözik. A kivételeket emuláció közben detektálja, a többi megszakítás pedig az éppen aktív utasítás elvégzése után kerül feldolgozásra.

A 16 bites módban futó operációs rendszerek és programok nagy része a BIOS rendszerhívásokat használja bizonyos I/O feladatok elvégzésére. Ez szükségessé teszi a BIOS rendszerhívások emulációját. Erre még azért is szükség van, mert nem valószínű, hogy a valódi gép BIOS-a ugyanazokat a beállításokat tartalmazza mint a virtuális gépé.

A 16 bites védett mód implementációja nem szükséges, mivel a ma haszálatban lévő operációs rendszerek (Windows, Linux, BSD, Darwin, stb.) egyike sem használja.

A 32- és 64 bites mód emulációja

Amikor az emulált kernel átlép védett módba (a PE bitet a CR0 regiszterben igazra állítja, majd ezután egy távoli ugrás (far jump) következik a 32 bites kódba), a virtuális gép a vezérlőstruktúrákat és táblázatokat a valódi processzorra másolja (bizonyos módosítások után), beállít a processzoron egy védett, compatibility módú kódszegmenst, majd átadja a vezérlést a virtuális gép kernelének.

Ha az emulált kernel valamilyen rendszerutasítást próbál elvégezni, a proceszor General Protection Fault kivételt generál. Ennek hatására a virtuális gép megpróbálja emuláli azt a bizonyos insrukciót, ügyelve a cím, port és egyéb átirányításokra és speciális esetekre. Ha emuláció közben is General Protection Faultot előidéző kódrészletet detektál az emulátor, akkor a kivételt átadja a virtuális gép kernelének, ugyanis ez azt jelenti, hogy a valódi gépen is kivétel jött volna létre.

A memóriakezelő emulációja az emulált kernel laptáblázatainak átdolgozásával és valódi processzorra való másolásával történik. Minden virtuális gép induláskor kap egy meghatározott mennyiségű, összefüggő fizikai memóriatartományt. Amikor a vituális gépen futó szoftver valamely, a valódi gépen nem létező laphoz próbál hozzáférni, az Page Fault kivételt generál. A laptáblázat elemeit frissítő INVLPG instrukció ugyancsak kivételt okoz. Page Fault érzékelésekor (ha azt nem a főkernel okozta) a virtuális gép meghatározza, hogy mi okozta a kivételt. Ha a kivételt olyan lap okozta, amely a virtuális gép címterében jelen van, a valódi gépében viszont nincs, akkor ezt a lapot elérhetővé teszi a valódi processzoron is. Első lépésként a virtuális gép laptáblázatainak kiértékelése által meghatározza a lap fizikai címét. Ezt a címet azután egy olyan egyedi címmé alakítja, amely nem ütközik a főkernel és a többi virtuális gép egyetlen más lapjával sem. Ez a virtuális gép számára kijelölt fizikai memóriatartomány és bizonyos címátirányítások alapján történik. Az így kapott egyedi címet azután a valós processzor laptáblázatába írja. Ha bizonyos okoknál fogva (pédául az átirányított cím nincs laphatárhoz igazítva, és a lap más-más részén más-más valódi laphoz kell hozzáférni) ez nem

kivitelezhető, akkor teljes emulációt alkalmaz. Ha az emulált gép kernele a fő laptáblázat báziscímét próbálja módosítani (a PML4 táblázatot, vagyis írja a CR2 regisztert), akkor az emulátor törli a valódi processzorról az összes olyan laptáblát, amely a virtuális géphez tartozik. Így a következő memóriahozzáféréskor létrejövő kivétel által az új lapok lesznek a valódi gép címterébe írva. A virtuális gép hasonlóan jár el az INVLPG instrukció hatására is, csak ezesetben a laptáblázatok megfelelő részét törli. Mivel ezek a műveletek igen sűrűn jelentkeznek, ezért szükséges egy olyan mechanizmus kifejlesztése, amely a laptáblázatok által használt memóriát igen gyorsan fel tudja szabadítani. Ennek az algoritmusnak az alapötlete azon alapszik, hogy a táblázat bizonyos részeinek törlésekor a törölt táblázat címét eltárolja, majd szükség esetén lépésenként szabadítja fel a felszabadított táblázat lapjait. Ha a virtuális gépen futó szoftver olyan címhez próbál hozzáférni, amelyet a főkernel is használ, akkor a szükséges instrukciót teljes emulációval végzi el, amely folyamán megtörténik a virtuális címek átirányítása a megfelelő fizikai címekre. Az esetben, ha a Page Fault létrejöttekor a lap létezik a valódi processzor címtartományában, vagy a virtuális gép címterében sem létezik, a kivétel továbbítódik a vrituális gép kernele felé.

A PCI elemek kezelése úgy történik, hogy a virtuális gép szkenneli a PCI buszt, és a talált hardverelemek adatait eltárolja. Amikor az emulált kernel szkennelni próbálja a buszt, az kivételeket okoz. E kivételek hatására végbemenő emuláció közben az emulátor ügyel bizonyos címekhez való hozzáférésre. Ilyen címeken (0xCF8 és 0xCFC) keresztül történik a PCI busz szkennelése is, ami lehetővé teszi az emulátor számára, hogy a valódi gépen detektált adatokat módosítsa. Így elrejthet bizonyos hardverelemeket, vagy hozzáadhat valamilyen emulált elemet. További figyelmet kell szentelni az IRQ konfliktusok elkerülésére, amelyek létrejöhetnek ha több emulált kernel ugyanazon az IRQ-t próbálja használni más-más célra. E probléma megoldásához figyelni kell a már használatban lévő IRQ-kat és olyanokat allokálni szükség esetén, amelyek még nincsenek használva. Ha ez nem lehetséges, akkor valamely más, nem aktív virtuális gép egy IRQ-ját kell ideiglenesen felszabadítani. Később, IRQ érzékelésekor a megszakításvektort és a verembe helyezett adatokat megfelelően módosítva az emulált kernel által eredetileg megadott IRQ-nak kell álcázni azt.

A főkernel nem használ DMA-t, abból az okból kifolyólag, hogy a DMA sávra az emulált kernelnek is szüksége lehet. Szintén rendkívül fontos, hogy a DMA használata az alsó 16Mb virtuális memória használatát igényli. Ez külön problémát vet fel: melyik virtuális gépé legyen az alsó 16 Mb. Az emulátor eltárol bizonyos információkat a virtuális gép címterének azon részeiről, amelyek az alsó 16Mb fizikai memóriára mutatnak. Ha az emulátor detektálja, hogy az emulált kernel programozni próbálja a DMA vezérlőt, allokál egy lapot a valódi memória alsó 16Mb-jában. Ezután az emulátor által a mappeléskor allokált memóra tartalmát az új lapra másolja, majd azt az újonnan lefoglalt memórialapra cseréli. Így a virtuális gép a valódi, alsó 16Mb-ban levő laphoz férhet hozzá. Az így allokált cím alapján programozza a főkernel a DMA sávot. Ha nincs elég rendelkezésre álló szabad memórialap az alsó 16Mb-ból, akkor egy másik, inaktív virtuális gép címteréből kell ideiglenesen eltávolítani a megfelelő lapot.

A virtuális gép hardverállapotának nyomonkövetésére is szükség van, a virtuális gép működésének gyors felfüggesztése érdekében (ez a gyors módszer csak akkor szükséges, ha schedulingkor ideiglenesen más virtuális gép válik aktívvá, és az használ valamely, az előző virtuális gép által használt hardverelemet is. Más szóval lehetővé teszi, hogy bizonyos hardverlemek minden pillanatban az aktív virtuális géphez legyenek hozzárendelve. Ez szintén nem szükséges, ha az emulátor csak az operációs rendszerek közötti gyors váltásra szolgál, azaz nem fut két operációs rendszer egy időben. Ilyen esetekben használható a BIOS egy értesítése, amely a laptopokon használatos a suspendelésre való parancs kiadására. Ez lényegesen lassabb az előzőnél, ezért schedulelésnél nem használható, viszont megvan az az előnye, hogy nem szükségesek hozzá meghajtóprogramok a főkernelben). Ezért szükséges bizonyos I//O címek és memóriatartományok adatátvitelének ellenőrzése. Az I/O címeknél ez közönséges ellenőrzéssel oldható meg: amikor a kivétel hatására elinduló emulátor bizonyos, előre meghatározott portot észlel, a megfelelő vezérlőnek (handlernek) adja át az irányítást, amely emulálja azt a bizonyos portot, és/vagy feljegyez bizonyos

adatokat. A memóriatartományoknál ez valamivel összetettebb. A laptáblázatok segítségével valósítható meg, úgy, hogy nem írjuk át a valódi processzor laptáblájába a figyelni kívánt címeket. Ekkor azokhoz való minden hozzáférés Page Fault kivételt okoz, amely alapján az emulátor végrehajthatja a címekhez hozzárendelt vezérlőket. E módszerrel szükség esetén a megfelelő hardver állapota nagyon gyorsan eltárolható. Ezt a módszert csak abban az esetben kell alkalmazni, ha a főkernel nem nyerhet ki elegendő információt közvetlenül a hardverelemből annak állapotának elmentésére.

III. ÖSSZEFOGLALÁS

E projekt alapot nyújt egy új típusú virtuális gép megalkotásához. A projektum tökéletesítése, a schedulelés és a virtuális gép működésének állaptának elmentése könnyen megvalósítható, viszont a projektum kidolgozásakor nem volt szükséges. Ezidáig elkészült a főkernel, valamint a virtuális gép fő részei. Ilyen a virtuális gép memóriakezelője, a memóriacímek átirányítása az emulált kernel által teljesen észrevehetetlenül, stb. A 16 bites emuláció működik. A védett mód implementálása folyamatban van, viszont majdnem kizárólag ugyanazokon, a már működő eljárásokon és módszereken alapul, amelyek már implementálva vannak (létre kell hozni az eljárásokat a szegmensek másolására és módosítására, a laptáblák szkennelésére, valamint el kell indítani az emulált kernelt felhasználói processzként). E projekt bizonytja, hogy a kernelek futtathatók felhasználói módban a teljes processzor emulációja nélkül, és hogy használhatják a számítógép valódi hardverelemeit korlátozás nélkül (a mai virtuális gépekkel ellentétben, amelyek minden hardverelemet emulálnak). E virtuális gép nagyban le fogja egyszerűsíteni több operációs rendszer használatát egy gépen. Nagyon hasznos olyan esetekben, amikor bizonyos hardverlemekhez nincs meghajtóprogram a felhasználó által használt operációs rendszer alá, és ez végett más operációs rendszert is használnia muszály (ismert például a szoftveri modemek problematikája Linux alatt). Arra is lehetőség nyílik, hogy egyik operációs rendszerből használjunk valamilyen, a másik operációs rendszeren működő hardverelemet (például a szoftveri modemek esetében összeköthetjük a virtuális gépeken egyidőben futó Windowsot és Linuxot virtuális hálózati kártyák segítségével, így lehetővé téve hogy a Linuxból is használható legyen az Internet). Bizonyos segédalklamazások használatával (amelyek továbbítják a kezelőfelületre vonatkozó információkat például egy, a virtuális gép által erre a célra létrehozott virtuális hardverelemen keresztül) az is elérhető, hogy az egyik operációs rendszerben futó alkalmazás a másik operációs rendszerből is használható, a megszokott és előnyben részesített környezet elhagyása nélkül. Hasznos azoknak a felhasználóknak is, akik operációs rendszer független programokat fejlesztenek, mivel a számítógép újraindítása nélkül tudják tesztelni programjaikat. A jövőbeni tervek között szerepel az emulált virtuális gép szoftverén történő hibakeresés (debuggolás) támogatása is. Ez nagy segítséget jelenthet a meghajtóprogramokat fejlesztő programozóknak is, mivel az operációs rendszertől függetlenül, gépi szinten követhetik nyomon a meghajtó programok és az egész operációs rendszer működését.

Legnagyobb újítás a mai virtuális gépekhez viszonyítva az, hogy a virtuális gép szoftvere majdnem teljes egészében a valódi processzoron fut. Ez nagy előrelépést jelent az emuláció sebességében. További újítás a valódi hardverelemek használatának támogatása. Ezzel lehetővé válik a kissebb fokú virtualizáció, ami azzal az előnnyel jár, hogy bizonyos hardverlemek teljes sebességükkel és képességeikkel használhatók (pl.: ez a videókártyáknál rendkívül sokat jelent). Nagy előny továbbá az is, hogy minden virtuális gép a számítógépnek azt a valódi részét használhatja, amit a felhasználó meghatároz, így nem csak egy operációs rendszer sajátítja ki az összes elemet, hanem azok feloszthatók a virtuális gépek között.

Legfőbb hiányossége ennek az új típusú virtuális gépnek, hogy nem képes az emulált kernel lapjait supervisor lapokként megjelölni. Ez azt jelenti, hogy a felhasználói programok is hozzáférhetnek a kernel privát memóriatartományaihoz. Mind ahogy már fentebb szó volt róla, ez legtöbb esetben nem jelent gondot, mivel az otthoni felhasználóknak, valamint a tesztelőknek és programozóknak akik a

virtuális gépeket használják általában nincs szükségük ezekre a védelmi mechanizmusokra. Ha viszont mindenáron szükséges a teljes védett mód, akkor ez az emulált operációs rendszer kernelének teljes emulációjával érhető el, ami viszont a virtuális gép sebességét nagymértékben csökkenti.

Hasonló projektumok a XEN (http://www.xensource.com) és a Qemu (http://fabrice.bellard.free.fr/qemu/). A XEN és ez a projektum között a legfőbb különbség, hogy az operációs rendszerek kernelét módosítani kell, hogy futhassanak a XEN-en. A XEN valójában nem egy valódi virtuális gép, mivel egy olyan virtuális környezetet hoz létre, amelyhez egy, a rendszer szinthez hasonló API-t biztosít. Ennek használatához a cél operációs rendszert módosítani kell. Ez nagy gond lehet a zárt forráskódú operációs rendszerek esetében, mint például a Windows. A Qemu képes bármely operációs rendszer futtatására, viszont nem képes a valódi hardverlemek használatára. Ezenfelül, mivel egy általános célú operációs rendszeren fut, sokkal nehezebben tud egyes dolgokat emulálni, és a hardvernek, és ezzel egyetemben a virtuális gépen futó szoftvernek sokkal jelentősebb hányadát emulálnia kell, ami ahhoz vezet, hogy az emuláció sokkal lassabb (http://fabrice.bellard.free.fr/qemu/qemu-tech.html).

Jövőbeni tervek:

- 32 bites védett módú operációs rendszerek támogatása. Könnyen megvalósítható a már megvalósított alapokon. Ki kell dolgozni a szegmensek másolására és átdolgozására szükséges algoritmust, az emulált kernel laptáblázatainak szkennelését, valamint a kernelt felhasználói módban kell futtatni.
- 64 bites operációs rendszerek támogatása. A 32 bites mód mechanizmusainak apró módosításaival kivitelezhető.
- A hardverelemek kezelése és virtuális hardverelemek támogatása. Ez lehetővé tenné a számítógép hardverelemeinek elosztását a virtuális gépek között.
- Az operációs rendszerek futásának felfüggesztése és a virtuális gép hardverállapotának elmentése. E problematika már a *A 32- és 64 bites mód emulációja* című fejezetben kifejtésre került.
- A teljes védett mód támogatása, mint választható opció. Ez azt jelenti hogy a kernel továbbra is használhatja a supervisor laptáblákat. (A *Nehézségek* című fejezetben kifejtésre került). Az emulált kernel kódjának teljes emulációjával valósítható meg.
- Több operációs rendszer egyidejű futásának támogatása. Itt jelentkezik a virtuális gép futásának gyors felfüggesztésének problematikája, amellyel a *A 32- és 64 bites mód emulációja* fejezet már foglalkozott.
- A virtuális gép állapotának elmentése a merevlemezre. Ez egy egyszerű bővítménye a virtuális gép futásának felfüggesztését végző mechanizmusnak.
- Hibakereső (debugger), amely lehetővé tenné az egész operációs rendszer futásának nyomonkövetését hardveri szinten. Ez nagy előnnyel jár a meghajtóprogramok és kernelek hibáinak kijavításakor.
- A Linux kernel közvetlen, fájlból való beolvasása (LILO vagy GRUB használata nélkül)

FELHASZNÁLT IRODALOM:

Könyvek:

- AMD, 2005: *AMD64 Technology: AMD64 Architecture Programmer's Manual Volume 2: System Programming*
- AMD, 2005: *AMD64 Technology: AMD64 Architecture Programmer's Manual Volume 2: General Purpose and system instructions*
- Paul A. Carter, 2003*: The PC assembly language*
- Randall Hyde*: The art of assembly language*

- Peter T. McLean: Information Technology – *AT Attachement with Packet Interface – 6 (ATA/ATAPI – 6)*
- Penn Brumm and Don Brumm, 1987: *80386 – A Programming and Design Handbook*

Weboldalak:

- Devlopment of a pmode multitasked disk driver <http://clicker.sourceforge.net/docs/teacup/Disk_Programming.html>
- Justin Deltener: *DMA programming*
- *Scheduling algorithms* <http://www.ctl.ua.edu/math103/scheduling/scheduling_algorithms.htm>
- *Job Scheduling Algoritms in Linux Virtual Server* <http://www.w3.org/TR/html14/strict.dtd>
- *The 'standard' cpu scheduling algorithm* <http://tunes.org/~unios/std-sched.html>