Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar

Kerékfy Miklós

MOBIL VIRTUALIZÁCIÓS MEGOLDÁSOK

Házi feladat a Virtualizációs technológiák és alkalmazásaik (VIMIAV89) című tárgyhoz

BUDAPEST, 2011

Tartalomjegyzék

1 Feladat ismertetése

A mobil készülékek világában megjelent virtualizációs technológiák érdekes új lehetőségeket adnak mind az egyszerű felhasználók, mind a fejlesztők, mind a céges IT vezetők számára. A virtualizáció több szempontból változtathatja meg a mobil telefonokról alkotott képünket:

- kiválthatja a biztonságos többmagos architektúrát, és így olcsóbb készülékeket tesz lehetővé
- akár több logikai telefonkészüléket használhatunk egy fizikai eszközön
- a magas költségű általános készülékeket felválthatják az alacsonyabb költségű alkalmazásorientált készülékek

Féléves feladatom a jelenleg elérhető technológiai megoldások megismerése, és azok összehasonlítása bizonyos paraméterek mentén.

2 Feladat pontosítása, célok meghatározása

A félév során megpróbálom felmérni annak lehetőségét, hogy egy valódi és működő mobil virtualizációs környezet milyen módon működhet, mik az előnyei és hátrányai a hasonló módszerekhez képest.

Ezen belül foglalkozom a VMware MVP (Mobile Virtualization Platform) vagy Horizon Mobile technológiájával, az Open Kernel Labs OKL4 megoldásával és érintőlegesen még a Xen on ARM rendszerrel.

3 Elvégzett munka

3.1 Virtualizáció a biztonság szempontjából

Az egyszerű mobil készülékeknél a biztonság azt jelentette, hogy a mobil használója ne tudjon belehallgatni mások beszélgetésébe vagy *adatfolyamába*, valamint hogy a felhasználó alkalmazott-e SIM kártyazárat. Manapság az „okostelefonok" világában sokkal több feladat merül fel. A korábban említetteken kívül a felhasználók által a készüléken tartott *tartalom* (telefonszámok, emailezés, képek/videók, stb.) valamint a különböző tartalomszolgáltatók fizetős szolgáltatásainak jogosultatlan elérése is ebbe a témába tartozik [1].

A jelenlegi mobil világban ezeket a problémákat meg lehet kerülni, ha a platform valamilyen szinten *kontrollált*, pl. csak Java alapú készülékek, vagy szigorúan kontrollált *zárt* operációs rendszerek, vagy akár két chipes architektúra a mobil rendszer teljes *fizikai* elválasztásához. Azonban ahogy egyre jobban elterjednek a nyílt vagy szabad szoftver alapú megoldások, ezekre a problémákra egyre sürgetőbb megoldást találni.

A világ vállalatainak szempontjából az okostelefon egy kihasználatlan *erőforrás*. Gyakorlatilag minden dolgozó zsebében van egy nagy számítási teljesítményű PC, mely a munkaidő nagy részében *kihasználatlanul* hever, hiszen a munkáltató IT osztálya biztonsági aggályok miatt nem engedi a céges adatforgalmat rajta használni. Mellette mindenki számára vásárolnia kell a cégnek készüléket, melyről telefonálhat és az email-jeit olvashatja. Ez a megoldás nem csak a cégnek, de az embereknek is rossz, hiszen ők pont azért vették az okostelefonjukat, hogy *használhassák*, és ezért ez állandó konfliktus forrása az IT és a többi munkavállaló között. Ennek a gyakorlatnak az utóbbi években megerősödő *BYOD* (Bring Your Own Device - Hozd a saját készüléked) mozgalom próbál véget vetni, mind munkáltatói, mind munkavállalói oldalról. Ennek a lényege az, hogy a felhasználók saját tulajdonú mobil készülékeit *bevonják* a céges környezetbe, alkalmassá téve azt mind a céges, mind a személyes kommunikációra.

Az így bevont készülékekkel kapcsolatban azonban olyan problémák merültek fel, melyekre a mobil virtualizáció nyújthatja az egyik legjobb megoldást. Ilyen például az, hogy el lehessen kerülni a céges és személyes telefonszámok *keveredését*; a ma használt MDM (Mobile Device Management - Mobil eszköz menedzsment)

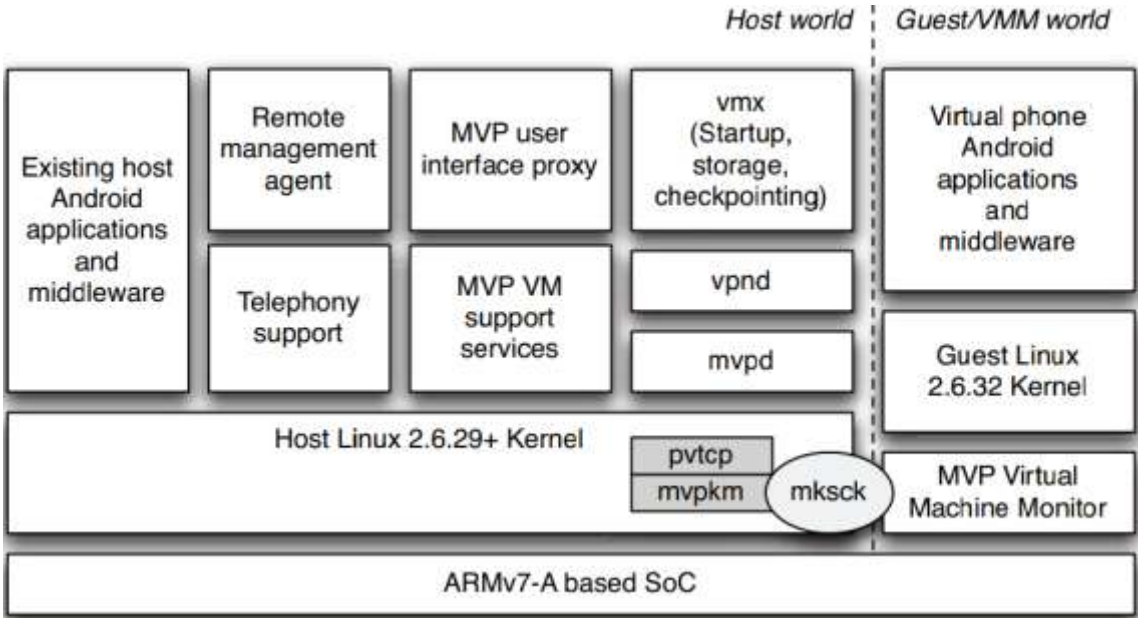
megoldások általában valamilyen *konténer* alapú megoldást nyújtanak, melynek azonban legfőbb hiányossága a rugalmatlan kezelhetősége, ugyanis minden telefonszám kereséskor be kell lépni az adott céges konténerbe annak érdekében, hogy a keresés a megfelelő helyen történjen. Ebből fakadóan sok felhasználó előbb-utóbb elkezdi átmásolni a személyes telefonkönyvébe a céges bejegyzéseit, hogy egy helyen érhesse el őket. Erre egy jól megvalósított virtualizációs rendszerben nincs szükség, mert a *kontextusváltás* gyors és egyértelmű lesz, ráadásul akár ugyanolyan vendég rendszer futhat akár céges és saját célra is, tovább könnyítve a felhasználók számára a kezelhetőséget. Továbbá a biztonság elérése érdekében sok olyan megoldást kell alkalmazni az MDM rendszerekben, melyek kényelmetlenné teszik a céges tartalom kezelését: külön, titkosítottan tárolt levélcsatolmányok, copy-paste funkció letiltása a levelezőben, stb. Ezeket szintén el lehet kerülni, hiszen a céges és saját rendszer egymástól teljesen függetlenül futhat akár *ugyanazon* a készüléken is.

3.2 VMware MVP

A VMware mobil virtualizációs megoldását MVP-nek nevezték el, azonban az utóbbi hónapokban a Horizon Mobile márkanév alá vonták be a megoldást. A továbbiakban az egyértelműség kedvéért MVP-nek fogom hívni magát a virtualizációs megoldást.

Az MVP-t elsősorban a VMware nyilvános publikációja [2] alapján ismerhettem meg. Az egész konstrukció arra a feltételezésre épül, hogy egy jól használható virtualizációs megoldásnak illeszkednie kell a BYOD elképzelésbe. Ezért olyan megoldást fejlesztettek, mely nem igényel a *telefon ROM-jába* égetett részeket, hanem csupán *kernelszintű* változtatásokat kell végigvinni. Ezentúl felmerült olyan igény is, hogy a hypervisor ne üljön rá feltétel nélkül minden *erőforrásra*, hanem legyen lehetőség arra, hogy egyes hardverelemek (pl. 3D grafika), melyek a céges környezetben feleslegesnek bizonyulnának, ne kerüljenek a befolyása alá. Megközelítésüknek továbbá előnye, hogy a gyártó által kevés speciális módosítást kell véghezvinni a készülékeken. A megoldás lényegében félúton áll a Type 1 és a Type 2 virtualizáció között, ugyanis bár nem szoftveres a virtualizáció, nem kell mégsem az egész készüléket a hypervisornak alárendelni.

A telefonba a gyártó által elhelyezett részt *mvpd*-nek (MVP Daemon) nevezik, mely lényegében egy *rootkit-hez* hasonlóan jogosultságokat ad a megfelelő folyamatok számára. Az így jogosultságokat szerzett mvpd betölti az *mvpkm* kernel modult (MVP Authenticated Kernel Module), mely képes átaadni az irányítást a kerneltől az MVP VMM-nek (Virtual Machine Monitor). A VMM átadja az irányítást a vendég operációs rendszernek, és tulajdonába veszi a *kivételkezelést* valamint a *lapozófájlokat*. Kivétel esetén, vagy egyéb szükséges esetben visszaadja a vezérlést a gazda rendszernek. A felépítés részletes diagramja az 1. ábraán látható.



1. ábra: Az MVP rendszer felépítése [2]

3.2.1 A processzor virtualizációja

A jelenlegi okostelefonok szinte kizárólag ARM architektúrára épülnek, ami külön figyelmet igényel a virtualizáció szempontjából. Az egyik alapvető különbség az ARM és x86 architektúrák között az, hogy az ARM-nek csak *egy felhasználó mód* szintje van. A másik fontos jelenség, melyre figyelemmel kell lenni, hogy vannak olyan utasítások, melyek végrehajtása során *nem jön létre trap*, azaz az ARM ISA (Instruction Set Architecture – utasításkészlet felépítés) nem kezeli le azokat; ezeket *érzékeny utasításoknak* nevezik.

Ezeket az alapvető tulajdonságokat figyelembe véve az MVP hibrid megoldása másképp oldja meg a különböző utasítások végrehajtását a virtuális kernelben. Alapvetően felhasználó módban fut a processzoron a kernel és a felhasználói szintű kód is, és az ISA-ra bízzák a privilegizált utasítások végrehajtását. Az érzékeny utasítások számára megoldást jelenthetne az erőforrás-igényes futási időben történő bináris fordítás, azonban ehelyett *paravirtualizációval* valósították meg. Ennek előnye, hogy kevésbé erőforrás-igényes, hátránya viszont, hogy a vendég operációs rendszer kernelét kell hozzá módosítani. Az VMWare megoldásában egy ún. LPV (Lightweight Paravirtualization – „könnyű” paravirtualizáció) nevű megoldást alkalmaznak. Ennek lényege egyrészt, hogy az operációs rendszernek csak az *architektúrafüggő* részét változtatják meg, másrészt, hogy ezeket a változtatásokat csak felszíni változtatásokként valósítják meg. Az érzékeny utasításokat vagy egy azonnali rendszerhívásra cserélik ki,

mely megvalósítja a trap-et, vagy helyére beillesztik a végrehajtandó kódrészletet. Ez utóbbi megoldás több munkával jár, azonban jobb teljesítményt nyújthat.

3.2.2 Memória virtualizáció

Mivel az ARM virtualizációhoz jelenleg nem létezik hardveres támogatás, a memória virtualizációt vagy egy *virtuális MMU* (Memory Management Unit – memóriamenedzsment egység) létrehozásával, vagy paravirtualizációval lehet megvalósítani. Utóbbi esetben viszont jóval nagyobb és összetettebb feladat lenne a vendég operációs rendszer kódjának átírása, hiszen nem csak néhány utasítást kéne kicserélni, hanem jóval nagyobb részeket, ezért az MVP megoldásában a virtuális MMU mellett döntöttek. A virtuális MMU lényegében egy *második indirekciót* vezet be a memóriacímzésnél; a szokványos logikai → fizikai leképezésen felül a VMM fenntart egy vendég fizikai → hoszt fizikai leképzést nyilvántartó táblát is.

A memória hatékony kihasználása érdekében fontos, hogy ne legyenek a hoszt és a vendég által sem használható memóriarészek. Ezért el kell kerülni azt a megoldást, hogy mindkettőnek valamilyen fix mérete van, melyet vagy kitölt, vagy nem; helyette a *ballooning* megoldást alkalmazzák, melynek lényege, hogy a rendelkezésre álló szabad erőforrást mind a hoszt, mind a vendég használatba veheti, ha arra szüksége van.

3.2.3 Tárhely virtualizáció

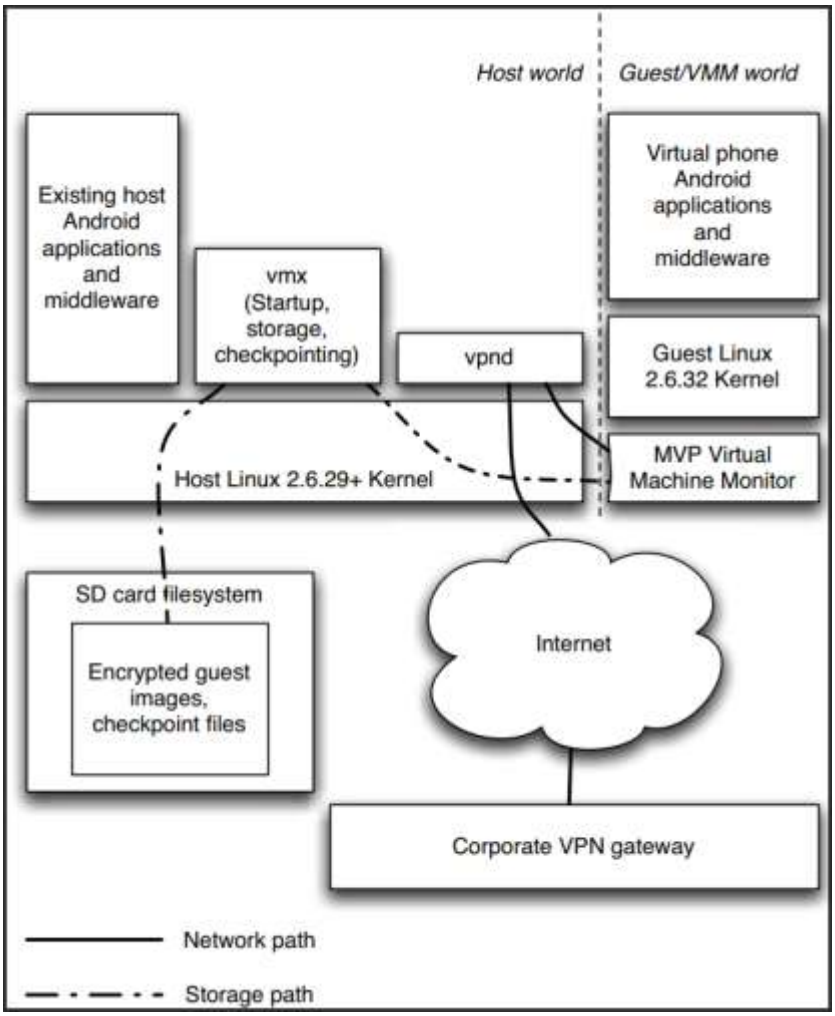
Az okostelefonok világában a tipikus tárhely két lépcsős: a készülék tartalmaz egy gyárilag beépített NAND flash memóriát, melynek magas ára miatt a mérete tipikusan 256 MB-tól néhány GB-ig terjed. Emellett általában bővíthető *microSD* kártyával, melynek mérete akár 32 GB is lehet. Mivel a NAND flash elérési sebessége gyorsabb, az MVP hypervisor oda kerül, azonban a virtuális image-ek nagy mérete miatt azokat érdemes a lassabb SD kártyán tárolni.

Hogy az SD kártyán tárolt virtuális gép sebessége elfogadható legyen, kihasználja a megoldás azt, hogy *szekvenciális írásoknál* jobban teljesít a kártya. Az MVP egy napló alapú struktúrában tárolja az egymás utáni I/O utasítások eredményét, és azt szekvenciálisan írja ki az SD kártyára.

A 3.2.6 Biztonság pont alatt tárgyalt SD kártya titkosítását a vmx modul végzi, a 2. ábraán bemutatott módon.

3.2.4 Hálózatkezelés

A hálózatkezelés klasszikus virtualizációja nagy számítási teljesítményt igényel, és növeli a csomagok által megtett utat. Ezt elkerülendő az MVP egy pvtcp modult (1. ábra) helyez a kernelbe, mely megvalósítja a kliens által küldött hálózati rendszerhívásokat. Ennek eredménye az, hogy nem kell egy virtuális hálózati interfészt kezelni, hanem a hoszt hálózati interfészét rögtön annak kerneljéből tudja a vendég operációs rendszer elérni. A 3.2.6 Biztonság pont alatt tárgyalt VPN elérést az mvpd-re épült vpnd elem biztosítja. A hálózati- és tárhely elérési utakat a 2. ábra mutatja be részletesen.



2. ábra: Hálózati- és tárhely elérési utak az MVP-ben [2]

3.2.5 Telefonálás

Az MVP mögött az az elképzelés húzódik meg, hogy a munkahelyi és a személyes telefont *teljesen* elválasszák. Ezért a javasolt megoldások szerint a készüléknek két különböző telefonszáma lenne, a két külön rendszer számára. Ennek egyik megoldása lehet két SIM (Subscriber Identifier Module) kártyával, vagy akár olyan SIM kártyával, melyhez két IMSI (International Mobile Subscriber Identity) tartozik. Másik megoldás lehet az, hogy az adott számokat egy vállalati PBX-en, (Private Branch eXchange) azaz egy privát telefonközponton végeztetjük, majd onnan valamilyen metaadattal tájékoztatjuk a készüléket, hogy melyik számra érkezik a hívás.

Lehetőség van az MVP integrálására a Unified Communications [3] rendszerekbe is, melyek segítségével a telefónia megoldása egy jóval magasabb minőségű szolgáltatásba épül be.

3.2.6 Biztonság

A biztonság kérdése az MVP esetében négy szempontból merül fel:

- A hoszt rendszertől kapott jogosultságok ellenére semmilyen alkalmazásnak ne legyen lehetősége, hogy a virtuális környezetbe bejusson, vagy abból *információhoz* férjen hozzá.
- A vendég rendszer hálózati forgalma egy adott céges VPN-nen keresztül menjen, azonban a hoszt rendszerből ez a kapcsolat *ne legyen elérhető*.
- Az SD kártyán tárolt virtuális gépek és minden azokhoz kötődő adat *blokkszinten titkosítva* legyen, hogy azt se a hoszt alkalmazásai által, se az SD kártyát eltávolítva más módon ne lehessen jogosulatlanul olvasni.
- A vendég környezetbe váltás *jelszóköteles* legyen bizonyos idejű inaktivitás után.

3.2.7 Kritika

A VMWare megoldását kritika elsősorban fő piaci konkurensüktől, az Open Kernel Labs-tól illette. Gernot Heiser, az OK Labs társalapítója több blogbejegyzésében kitér az MVP általa feltárt hiányosságaira. Ezek között van:

- A sebessége. A VMWare által bemutatott MVP rendszernek az Android Developer Phone (ADP1) készüléken futó változata a bemutató tartó Brad Suessmith által is bevallottan lassú, ami Heiser szerint azt mutatja,

hogy az MVP *indokolatlanul lassúvá* teszi az egyébként elég gyors ADP1 készüléket. [4]

- A virtualizáció módja. Az MVP hibrid mivolta abban rejlik, hogy a hypervisor nem az OS alatt (Type 1), nem az OS felett (Type 2) hanem az OS-en belül fut. Ebben Heiser szerint az a probléma, hogy a BYOD szempontjából *előnnyel nem szolgál*, hiszen az mvpkm modult a gyártónak kell elhelyeznie, viszont a virtuális gépnek meg kell bíznia az OS-ben. Ha tehát a rendszert megtörik (Android esetében ez nagyon könnyű), a virtuális gépet is irányíthatják. Ezek mellett még azt a gondolatot is meglobogtatja, hogy egy rosszindulatú hacker akár direkt magát az mvpkm-nek *álcázó kódot* írhat egy ismerten MVP-támogatott készülékre, ezzel további támadásoknak kitéve azt. [5]
- A biztonság. A korábban említett biztonsági problémák gyakorlatilag érvénytelenítik a VMWare összes próbálkozását arra, hogy biztonságossá tegyék magát a virtuális gépet. Az SD kártya titkosítása csak az elvesztett készülék esetén ér valamit, ugyanis a megtört operációs rendszerben a rosszindulatú programok a *memóriából* kiolvashatják az éppen olvasott titkosított adatot. Hasonló módon a VPN sem ér semmit, hiszen afelett is átveheti az irányítást egy rosszindulatú program. [6][7]

3.3 OKL4 microvisor

Az Open Kernel Labs OKL4 nevű megoldása az L4 mikrokernelek családjára épül. Mivel a megoldás egyfajta ötvözése a mikrokernel-nek és a hypervisor-nak, a *microvisor* elnevezést alkalmazzák rá. [8] A microvisor fő tulajdonsága, hogy amíg megtartja a mikrokernel elődjeinek kompaktságát és hordozhatóságát, a legjobb hypervisor-ok virtualizációs teljesítményével bír.

Az OKL4 a klasszikus *Type 1 virtualizáció*, tehát a hardver teljesen a microvisor irányítás alatt áll, és a microvisor nyújt virtuális eszközöket a vendég OS-ek számára. A virtuális eszközök az OKL4 microvisor esetében [9]:

- Virtuális CPU-k (*vCPU*), melyeken a mikrokernelekben megszokottól eltérően nem támogatott a több szál létrehozása, azt ugyanis a vendég OS-re bízza. Több vCPU-ra csak akkor van szükség, ha a vendég számára több fizikai CPU-t szeretnénk adni.
- Virtuális MMU (*vMMU*), melyet a vendég az MVP-ben bemutatott móddal megegyezően tud használni.
- Virtuális eszközregiszterek és megszakítások (*vIRQ*) az I/O műveleteket valósítják meg.
- Kommunikációra a vIRQ-k mellett *csatornák* állnak rendelkezésre. Ezek olyan kétirányú FIFO (First In First Out) sorok, melyek hossza felhasználói módban konfigurálható.

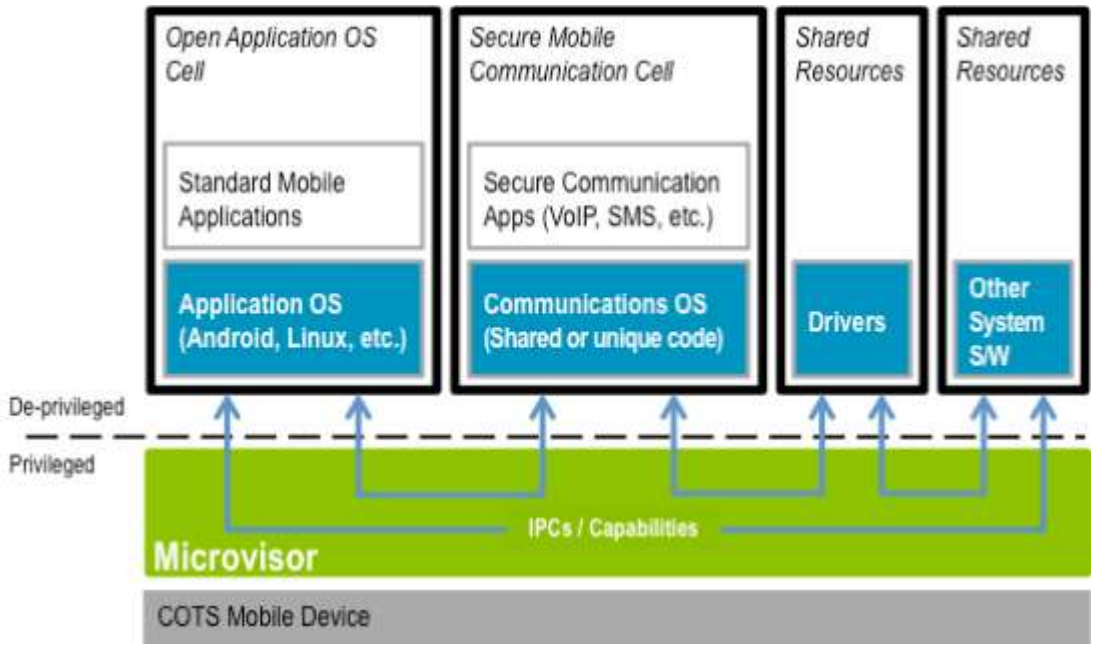
3.3.1 OKL4 működése a VMWare MVP-hez hasonlítva

Az OKL4 alapvető előnye az MVP-hez képest annak Type 1 mivolta. Minden fajta operációs rendszer esetén használható megoldást tud nyújtani, míg az MVP kizárólag Android alapú hosztra készült el, és új rendszerre való elkészítése sok munkába kerülne. Az OKL4 működése nem korlátozza azt le két darab elkülönített virtuális készülékre, ugyanis csak a fizikai erőforrások mennyiségétől függ a vendégek lehetséges száma. Mivel az OKL4 megvalósít egy egyszerű Posix interfészt is, lehetőség van arra, hogy natív módon, vendég *OS nélkül* fusson rajta program. Emiatt az OKL4 elnevezési szintaktikájában nem vendégekről, hanem *cellákról* beszélünk, mely egy-egy elkülönített környezetet jelent.

Míg az MVP esetében a paravirtualizáció a hoszt kódjának jelentősebb átírását jelenti, az OKL4 tulajdonképpen egy új architektúrát jelent, amelyre *portolni* kell az

adott rendszert; ez Linux esetében nagyjából egy 20 sor kódot tartalmazó patch formájában megvalósítható. [9]

Az OKL4 előnye továbbá a *gyors IPC-k* (Inter Process Call, folyamatközti hívások) mikrokernelből örökölt volta, valamint az, hogy *valós idejű* alkalmazásokra optimalizáltan készült a kód. A 3. ábraán egy kommunikációs diagram szerepel, mely bemutatja azt az alapszituációt, melyre az OKL4 alkalmazása a leginkább megfelelő. A készüléken lévő Android vagy Linux rendszer mellett egy *külön cellában* van a biztonságos kommunikációt biztosító rendszer, valamint külön-külön cellákban lehetnek a két rendszer által megosztottan használt *erőforrások* (meghajtók, célszoftverek) is.



3. ábra: OKL4 kommunikációs diagram [10]

Mivel az IPC-k konkrétan meghatározott hívások, nem tud a nyílt rendszeren jogosultságokat illetéktelenül szerzett kód beleavatkozni a biztonságos kommunikációba. Ezzel az architektúrával olyan biztonsági követelményeknek is meg tud felelni a rendszer, melyek kormányügynökségeknek vagy paranoiás óriáscégeknek vannak.

3.3.2 OKL4 és a Xen on ARM

A Xen egy rendkívül jó és megbízható virtualizációs platform enterprise környezetben, ezért triviális gondolat volt az, hogy ARM architektúrára is portolják. Ezt a Samsung mérnökei 2008-ban publikált módszerükkel meg is valósították [11]. A megoldás azonban a vártakkal ellentétben gyenge teljesítményt nyújtott. Heiser [9] megállapított néhány olyan szempontot, mely alapján a Xen nem tudja soha nyújtani azt a teljesítményt, melyet az OKL4.

Mivel a Xen, más enterprise hypervisor-okhoz hasonlóan nem valós idejű működésre lett tervezve, a mobil telefonokban elvárt működésük nem lehet elég jó. A tapasztalat azt mutatja, hogy utólag áttervezni egy kernelt valós idejű működésre nagyon nehéz és rendkívül kétséges eredményekkel jár. Bár a Linux esetében sok év alatt sikerült némi valós idejű működést elérni, komoly alkalmazásokra nem használják. Hasonlóan a gyors kommunikáció elengedhetetlen; a Xen virtuális hálózatra alapuló rendszere túl lassú és erőforrásigényes, a gyors és alacsony erőforrásigényes IPC-k sokkal jobb teljesítményt tudnak nyújtani.

Memória szempontjából is rendkívül erőforrásigényes a Xen, ami szintén hátrányos, ugyanis a mobil eszközök esetében a memóriák fogyasztása jelentősen csökkentheti az akkumulátoridőt. Maga a Xen forráskódja is legalább egy nagyságrenddel nagyobb az OKL4-nél, aminek következménye lehet, hogy hibák is könnyebben fordulnak elő benne.

A teljesítmény analízisét a [8][9][11] publikációkban közölt mérések adatai alapján készíthetjük el. A Xen teljesítményét az lmbench programmal mérve a 4. ábraán látható adatokat kapták a Samsung mérnökei. Bal oldalon a natív Linux-on futtatott parancsok sebessége, középen a Xen alatt futó paravirtualizált Linux-é, jobb oldalon pedig a kettő sebesség aránya áll.

Latencies measured in microseconds			
Tests	Native	Paravirt	Ratio
lat_pipe	135.13	234.42	1.735
fork+exit	2891.75	10021.0	3.465
fork+execve	3109.25	10524	3.385
SysV semaphore	45.974	81.42	1.77
lat_unix	251.41	431.85	1.70
signal handler	11.23	20.43	1.82
null syscall	1.13	2.83	2.50
read syscall	2.60	4.94	1.90
write syscall	2.25	4.16	1.85

4. ábra: Xen teljesítménye az lmbench programmal mérve [11]

Az 5. ábra bemutatja a korábban mért sebességek összehasonlítását OKL4 és Xen esetén. Látható, hogy majdnem minden esetben az OKL4 teljesít jobban, ám ahol a Xen, az leginkább az OKL4 esetén alkalmazott viszonylag alacsony szinten beavatkozó paravirtualizáció miatt van. Ahol az OKL4 a natív Linuxnál jobban teljesített, ott a kontextusváltás sebessége jóval nagyobb, ugyanis az OKL4 nem üríti ki a cache táblákat. Ez a sebességnövekedés egy példája annak, hogy az egy nagyságrenddel kisebb méretű kódot mennyivel könnyebb jól optimalizálni.

	Latencies [μs]		Rel. Perf.	
Benchmark	Native	Virt./OKL4	OKL4	Xen
pipe	756.59	84.84	8.92	0.58
fork	6469	8742	0.74	0.29
fork+exec	59715	75515	0.79	0.30
semaphore	261.6	21.08	12.41	0.56
unix	1292.2	115.01	11.24	0.59
signal handler	14.26	54.76	0.26	0.55
null syscall	1.14	5.40	0.21	0.40
read syscall	3.34	8.45	0.40	0.52

5. ábra: OKL4 és Xen teljesítménye az lmbench programmal mérve [9]

4 Összefoglalás

A félév során megismert technológiák több alapvető koncepciót valósítanak meg. Jelenleg még egyik megoldást sem lehet megtalálni egy átlagos mobil készülékben, és amíg meg nem jelennek az adott virtualizációs megoldást támogató készülékek, gyakorlatban nehéz elemezni a megoldások egymáshoz való viszonyát.

Megállapítható viszont, hogy Type 2 virtualizációs megoldás a gyakorlatban nem létezik mobiltelefonokra, aminek főleg az az oka, hogy még alacsony a teljesítményük a készülékeknek. A jelenleg rendelkezésre álló technológiák közül a legérdekesebb és leginkább előremutató az Open Kernel Labs megoldása, mivel az ő megoldásuk sikeresen elkerül olyan csapdákat, melyekbe a más architektúrák feletti virtualizációban gyakorlott cégek beleesnek.

Irodalomjegyzék

[1] J. Brakensiek, A. Dröge, M. Borreck, H. Härtig, A. Lackorzynski: Virtualization as an Enabler for Security in Mobile Devices, In: IIES '08 Proceedings of the 1st workshop on Isolation and integration in embedded systems, 2008., pp 17-22

[2] K. Barr, P. Bungale, S. Deasy, V. Gyuris, P. Hung, C. Newell, H. Tuch, B. Zoppis: The VMware Mobile Virtualization Platform: is that a hypervisor in your pocket?, In: ACM SIGOPS Operating Systems Review, Vol. 44 No. 4, 2010., pp 124-135

[3] B. Elliot, S. Blood: Magic quadrant for Unified Communications, In: Gartner RAS Core Research Note , G00201349 (2010)

[4] G. Heiser: Fastest and most secure? In your dreams!, Forrás: http://www.ok-labs.com/blog/entry/safest-and-fastest-in-your-dreams/, Letöltés: 2011.10.26

[5] G. Heiser: VMware MVP: What it really is, Forrás: http://www.ok-labs.com/blog/entry/vmware-mvp-how-it-works/, Letöltés: 2011.11.06

[6] G. Heiser: VMware's MVP—Encryption Doesn't Make It Secure!, Forrás: http://www.ok-labs.com/blog/entry/vmwares-mvpencryption-doesnt-make-it-secure/, Letöltés: 2011.11.03

[7] G. Heiser: Hey VMware: Secure It Ain't!, Forrás: http://www.ok-labs.com/blog/entry/hey-vmware-secure-it-aint/, Letöltés: 2011.11.09

[8] G. Heiser, B. Leslie: The OKL4 Microvisor: Convergence Point of Microkernels and Hypervisors, In: APSys '10 Proceedings of the first ACM asia-pacific workshop on Workshop on system, 2010., pp 19-24

[9] G. Heiser: Hypervisors for Consumer Electronics, In: Consumer Communications and Networking Conference, 2009. CCNC 2009. 6th IEEE, pp 1-5

[10] R. McCammon: Technology Whitepaper SecureIT Mobile, 2010, Forrás: http://www.ok-labs.com/landing/secure-it-mobile/, Letöltés: 2011.11.02

[11] J.-Y. Hwang, S.-b. Suh, S.-K. Heo, C.-J. Park, J.-M. Ryu, S.-Y. Park, and C.-R. Kim: Xen on ARM: System virtualization using Xen hypervisor for ARM-based secure mobile phones. In5th IEEE Consumer Comm. & Networking Conf., pp 257–261, Las Vegas, NV, USA, Jan 2008.