Budapesti Műszaki és Gazdaságtudományi Egyetem
Virtualizációs technológiák és alkalmazásaik (VIMIAV89)

User-mode Linux Kernel

Nyebehaj Gergő
2009.12.08

User-mode Linux Kernel

Tartalom:

1. Feladatismertetés

A tárgy honlapján lévő feladatkiírásban ennyi szerepelt: „User Mode Linux: virtuális Linux futtatása Linux felett”. Ám mivel egy Linux kernel futtatásának önmagában nem sok értelme van, valamint nem is különösebben látványos, úgy döntöttem, mélyebben fogok foglalkozni a témával: bemutatom, hogyan lehet felette grafikus felületet és egyéb szoftvereket működtetni, hogyan lehet a használatát kényelmesebbé, felhasználóbarátabbá tenni, valamint megvizsgálom, milyen teljesítményt képes nyújtani ez a megoldás a fizikai gépen futó host kernelhez és más platformvirtualizációs szoftverekhez képest.
A feladat megoldása során forrásként a projekt honlapja (http://user-mode-linux.sourceforge.net/) valamint különböző levelezőlisták álltak rendelkezésemre. Ezekből csak technikai információkat sikerült szereznem, a felhasznált scripteket saját magamnak kellett megírnom.

2. Rövid áttekintés

A User-mode Linux a Linux kernel egy portja, mely fizikai processzor helyett egy másik, a hardveren már futó Linux kernel felett képes működni megszokott, normál folyamatként.
Eredetileg egy patchként volt elérhető a 2.2.x kernelverziókhoz, ám a 2.6.x verzióktól kezdve beolvasztásra került a hivatalos kernel forrásfába. Jelenleg nem túl széleskörűen alkalmazható, hiszen csak az x86 és az x86_64 architektúrájú processzorokat támogatja, de már folyik a munka az egyéb architektúrákra (IA-64, PowerPC) való portolásán is.

Habár régen szívesen alkalmazták biztonságos, a host rendszertől szeparált konténerek kialakítására, valamint – mivel a host és a guest kernel verziója eltérhet – kerneltesztelési és fejlesztési célokra, ma már háttérbe szorult a jellemzően jobb teljesítményű kereskedelmi megoldásokkal, valamint a konténerrendszerekkel – például OpenVZ – szemben.
A User-mode Linux egy teljes egészében paravirtualizált megoldás. A kernel pontosan „tudja”, hogy nem valódi hardver felett fut, sőt, amennyiben az rendelkezik ilyenekkel, a host kernel speciális virtualizációs szolgáltatásait is képes kihasználni. A forrásfában külön architektúraként van jelen, a host kernel rendszerhívásait használja, így nincs szükség tényleges hardvereszközök emulációjára.

Habár tudás tekintetében alig marad el nagyobb versenytársaitól (Vmware, VirtualBox, stb.), mint a linuxos megoldások általában, egyáltalán nem felhasználóbarát. Grafikus felülettel nem rendelkezik, a beállításait parancssori paraméterekként veszi át, melyekből meglehetősen hosszú és bonyolult kombinációkra lesz szükségünk, ha egy gyakorlatban is használható rendszert szeretnénk kapni.

Az UML főbb képességei:

- blokkos eszközök emulálása
- virtuális hangeszköz (OSS)
- virtuális hálózati eszközök
- saját ütemező és memóriakezelő
- soros konzolok (a host rendszerről elérhetők) és soros portok emulációja
- fizikai USB eszközökhöz való hozzáférés
- fizikai PCI eszközökhöz való részleges hozzáférés
- emulált eszközök (például hálózati kártyák) futás közbeni csatlakoztatása és eltávolítása
- a rendszer futásának átmeneti felfüggesztése

A lista ugyan nem tűnik túl hosszúnak, de kiegészítve a host rendszer által nyújtott szolgáltatásokkal (például hálózati hidak, iptables, megfelelő fájlrendszer esetén sparse fájlok) már egészen komoly dogokat lehet megvalósítani egy User-mode Linux Kernel-lel.

A továbbiakban egy ilyen, a kereskedelmi platformvirtualizációs szoftverekre funkciók tekintetében minél jobban hasonlító rendszer összeállítását fogom leírni.

3. Egy működő UML rendszer összeállítása

Hogyan kezdjünk hozzá?

Amire szükségünk lesz: maga a kernel és egy fájlrendszer, amit majd használhat.
Ezek beszerzésére két lehetőségünk van. Letölthetünk belőlük előre legyártott példányokat a projekt weblapjáról (http://user-mode-linux.sourceforge.net/), vagy elkészíthetjük őket magunk is.
Mivel a fenti címről letölthető kernel nem igazán friss, valamint javasolt a host kernel verziójával megegyező verziójú guest kernelt futtatni, én az utóbbi lehetőséget választottam.
A host rendszerem eredetileg egy 9.04-es Ubuntu Linux. Mivel egy ilyen kernel fordításához pont ugyanazokra az eszközökre van szükség, mint egy fizikai gépre szánthoz, a következő csomagok telepítése javasolt:

```
kernel-package
linux-source
```

Ekkor a /usr/src mappába bekerül egy linux-source-2.x.xx.tar.bz2 nevű fájl. Ez az én rendszeremen konkrétan linux-source-2.6.30.tar.bz2 volt.
Kicsomagolás után létrejön egy új könyvtár, benne a teljes kernelforrással.
User módú kernel fordításához a konfiguráló- és fordítóeszközöknek tudniuk kell, hogy milyen architektúrára dolgozzanak. Ennek beállítására a legegyszerűbb megoldás az ARCH környezeti változó értékének megfelelő, jelen esetben „um”-re való beállítása.

```
~$ export ARCH=um
```

Ez után már a kernel fordítása a megszokott módon folytatható.

```
~$ make defconfig
~$ make menuconfig
```

Utóbbi parancs hatására elindul a jó öreg ncurses alapú kernelkonfigurációs eszközünk. Itt én két említésre méltó változtatást hajtottam végre:

1. Az UML-specific options menüben a Host processor type and features alatt a Processor family értékét átállítottam Core2 / Newer Xenon-ra. (Természetesen ez a célhardvertől függ)
2. A Kernel hacking menüben kikapcsoltam a Kernel debugging-ot. Ezzel nagyjából az egyhatodára csökkenthető a lefordított kernel bináris mérete.

Attól függően persze, hogy milyen célra szánjuk a kernelünket, szükség lehet még más beállítások megváltoztatására is. Például alapértelmezetten ki van kapcsolva a FUSE és az NFS (mind kliens-, mind szerveroldali) támogatás, valamint a linuxon kevésbé használatos fájlrendszerek (pl. NTFS, FAT) kezelése.

A konfigurálás befejezése után a kernel a make paranccsal fordítható. Ennek hatására a forrás könyvtárában létrejön egy „linux” és egy „vmlinuz” nevű bináris állomány. Ezek bájtra pontosan megegyeznek, ezért a továbbiakban bármelyik használható közülük.

Hátravan még a fájlrendszer. Az UML kernel elvileg ebből kétfélét tud kezelni. Egyrészt egy komplett RAW lemezképet, blokkos eszközként, másrészt egy egyetlen partíció másolatát, tehát egy fájlrendszert tartalmazó képet. Tapasztalataim szerint bootolni csak az utóbbiról képes, másodlagos eszközként viszont már használhatjuk bármelyiket. Természetesen a kernelbe bele kell fordítani az adott fájlrendszer kezelésének képességét.

Ennek a fájlrendszernek az előállítására megint több lehetőség kínálkozik. Létrehozhatunk például egy üres fájlt, benne egy fájlrendszert mke2fs-el, felcsatolhatjuk loop eszközként és rámásolhatjuk egy telepített linux rendszer fájljait. Ez a következőképpen néz ki:

```
~$ dd if=/dev/zero of=uml.img bs=1024 seek=3145728 count=1
~$ mkfs.ext3 uml.img
~$ mount -o loop uml.img /media/umlmnt
~$ cp -r /media/masolandorendszer/* /media/umlmnt --preserve
~$ umount /media/umlmnt
```

Én egy másik megoldást választottam. Qemu virtuális gépben feltelepítettem egy Ubuntu Linuxot. Azért esett pont erre az eszközre a választásom, mert saját formátum helyett képes RAW fájlokat használni merevlemezek emulálására. A telepítés utáni RAW képből pedig a dd skip= megfelelő paraméterezésével kimásoltam a komplett partíciót, amin a gyökérfájlrendszer volt. Ez még mindig egy bonyolult megoldás, mivel a qemu parancssora sem kifejezetten nevezhető felhasználóbarátnak, ahogy az sem, hogy az

```
~$ fdisk -l /qemuimage.bin
```

kimenetéből az ember kibányássza a megfelelő méreteket, és ez alapján kitalálja, hogyan kell a dd-t paraméterezni.

A legegyszerűbb bár legidőigényesebb módszer, hogy feltelepítünk például VirtualBox alá – vagy akár egy fizikai gépre – egy Linux disztribúciót, beállítjuk a hálózatot, majd ezen a gépen egy Live CD-ről bootolva a dd + netcat párossal lemásoljuk a megfelelő partíció tartalmát. Ehhez a következő parancsokra lesz szükség, feltéve, hogy a host gép IP címe 192.168.0.10, a forrásként szolgáló rendszeren pedig a telepített Linux disztribúció gyökere a /dev/sda1 partíción található:
A host gépen:

```
~$ nc -l -p 13772 > /uml.img
```

A frissen telepített rendszeren pedig:

```
~$ dd if=/dev/sda1 bs=1024 | nc 192.168.0.10 13772
```

A partíció méretétől függően ez eltarthat egy darabig, de ha kész, megint csak egy UML-el használható fájlrendszer lesz az eredmény.

Itt jegyezném meg, hogy különböző levelezőlistákon láttam megoldásokat, hogyan lehet például egy debian telepítőt rávenni, hogy működjön egy UML kernel felett, de ezek a nekem sokkal bonyolultabbnak tűntek, mint az, hogy „kézzel” elkészítsem a fájlrendszert.

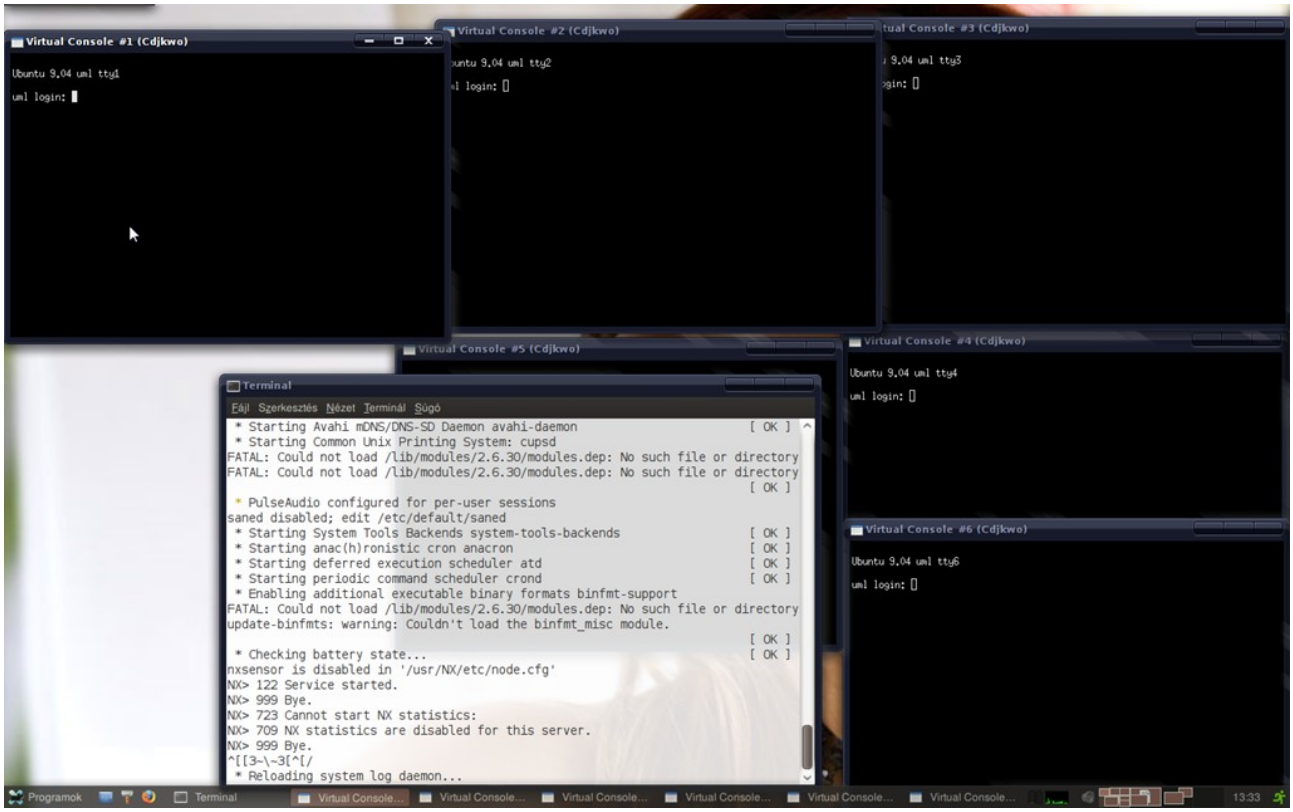
Már eddig eljutni is meglehetősen bonyolult és nehézkes volt, és még csak ekkor tartottam ott, hogy megpróbálhattam elindítani a kernelemet.

```
~$ ./linux ubda=uml.img mem=512M
```

Természetesen semmi sem működhet elsőre. A fájlrendszerkép nem lehet olyan partíción, ami FUSE használatával lett felcsatolva, így nem működik smbfs és ntfs-3g fájlrendszerről sem.

A bootolás során abban a terminálablakban, amelyikből indítottam az UML kernelt, jelentek meg azok az adatok, amiknek fizikai gép esetén a képernyőn kellett volna, valamint egy felbukkanó xterm ablakban egy hibaüzenet, miszerint nem található a rendszeremen egy bizonyos port-helper nevű program.
Némi keresgélés után kiderült, hogy ez az uml-utilities csomag része, így annak telepítése után végre eljutottam egy sikeres bootolásig.

A boot folyamat során az indításra használt terminálablakon kívül megnyílik még 6 xterm példány. Ezek az UML kernel virtuális konzoljainak elérésére szolgálnak (tty1-tty6), rajtuk keresztül lehet bejelentkezni a rendszerbe.
Az egész így néz ki egy képernyőképen:



1. ábra: Az UML kernel elindítása során megjelenő ablakok

Az UML kernel természetesen képes elindulni grafikus felület nélkül is, ekkor csak egy rövid hibaüzenet jelenik meg, ami arról tájékoztat minket, hogy az xterm folyamatok indítása sikertelen volt.

Kicsit nézelődtem a rendszerben. Érdekes látvány a top, amint 100% CPU üresjáratot mutat (natív Linuxon ez ritka eset), valamint a

```
~$ cat /proc/cpuinfo
```

kimenete is. Ez az én rendszeremen:

```
processor       : 0
vendor_id       : User Mode Linux
model name      : UML
mode            : skas
host            : Linux lapi 2.6.30 #1 SMP Sun Jun 14 00:51:48 CEST 2009 x86_64
bogomips        : 496.43
```

A mode: skas sor jelzi, hogy a host kernel is fel van rá készítve, hogy felette egy UML kernel fog futni. Ennek hatására a virtuális rendszer gyorsabban működik, mint a skas (Separated Kernel Adress Space) módot nem támogató kernelek felett, valamint kevesebb memóriát használ. Sőt, ez a futási mód még biztonsági problémákat is megold, hiszen az ún. „tt mód”-ban (Tracing Thread) futtatott UML kernelek memóriaterületére a felettük futó folyamatok akár írhatnak is (!).

Az eddig összeállított rendszer most már egészen működőképesnek tűnt. Az egyetlen problémám az volt, hogy nem volt hálózati eszközöm, nem „láttam ki” a virtuális gépből. Így a következő megoldandó feladat ennek beállítása lett. Ehhez először is leállítottam a guest rendszert.

```
~$ shutdown now -h
```

Az UML többféle hálózati csatlakozást biztosít. Létrehozhatunk vele virtuális switchet, melyet maga kezel a különböző UML kernelek közt, de rábízhatjuk ezt a feladatot egy a host rendszeren futó daemonra is a jobb teljesítmény érdekében. Használhatunk virtuális hálózati interfészt a hoston,

amely az UML kernel belső hálózati kártyájával áll közvetlen kapcsolatban, illetve képes NAT-szerű kapcsolat megvalósítására is, ha esetleg nem rendelkeznénk root jogokkal a host gépen.

(Van az UML virtuális hálózatának még egy érdekes működési módja, ami a host forgalmát gyűjti és irányítja tovább egy belső csatolóra. Ez például a host hálózati forgalmának megfigyelésére használható fel, és kizárólag egyirányú kommunikációra alkalmas).

Mivel én azt szerettem volna, hogy a rendszerem rendelkezzen internetkapcsolattal és a fizikai hálózatomat is elérje, nem csak IP protokollal, a második lehetőséget választottam. Ehhez először is be kellett állítanom a host oldalon egy tap adaptert. A TUN/TAP adapterekhez a Linux kernelekben (modul formájában) beépített támogatás van, így ez nem túl bonyolult feladat:

```
~$ tunctl -u 1000
```

ahol az 1000-es szám helyére az UML-t futtató felhasználó uid-jét kell beírni.
Az UML-nek szüksége van továbbá írási jogosultságokra a /dev/net/tun eszközre. Az egyszerűség kedvéért ezt én most minden felhasználónak megadtam, majd felélesztettem a létrejött tap0 adaptert:

```
~$ chmod a+rw /dev/net/tun
~$ ifconfig tap0 192.168.0.254 up
```

Ezek után az UML-t a következő parancssorral indítottam:

```
~$ ./linux ubda=uml.img mem=512M eth0=tuntap,tap0
```

Bootolás után az ifconfig eth0 parancs a következő kimenetet adta:

```
eth0   Link encap:Ethernet  HWaddr a6:dc:9c:d3:e6:5b
       inet addr:192.168.0.222  Bcast:192.168.0.255  Mask:255.255.255.0
       UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
       RX packets:110 errors:0 dropped:0 overruns:0 frame:0
       TX packets:102 errors:0 dropped:0 overruns:0 carrier:0
       collisions:0 txqueuelen:1000
       RX bytes:9548 (9.5 KB)  TX bytes:12533 (12.5 KB)
       Interrupt:5
```

A host oldalon még meg kellett oldanom, hogy minden csomag eljusson a rendeltetési helyére.. Erre két módszer létezik:

1. A tap0 interfészt hálózati hídba lehet kötni a fizikai eth0 csatolóval
2. Rá lehet bírni a host kernelt a hálózati csomagok megfelelő irányba való továbbítására

Mivel a második megoldás egyszerűbb, és egyelőre nem akartam kihasználni a hálózati híd által nyújtott előnyöket, ennél maradtam:

```
~$ echo 1 > /proc/sys/net/ipv4/ip_forward
~$ route add -host 192.168.0.222 dev tap0
~$ echo 1 > /proc/sys/net/ipv4/conf/tap0/proxy_arp
~$ arp -Ds 192.168.0.222 eth0 pub
```

Ezek után az UML kernel pingelhető lett, valamint a neki szánt fájlrendszer előkészítésekor feltelepített OpenSSH server is elérhetővé vált, így innentől kezdve kényelmi okokból ezt használtam a parancssoros eléréshez.

4. A rendszer kényelmesebbé tétele

Mivel meglehetősen kényelmetlen lenne ezeket a parancsokat a rendszer minden indításakor begépelni, ezért összeállítottam egy egyszerű scriptet, ami elvégzi a szükséges műveleteket és elindítja a kernelt (terminal.sh). Ennek bootolására 35 másodpercet vár, majd elvégzi a bootolás után még hátralévő feladatokat.
Azért, hogy a grafikus felületről is kényelmesen el lehessen indítani, ezt a scriptet egy másik script hívja meg egy gnome-terminalon keresztül (uml-start.sh).

terminal.sh

```
#!/bin/bash

konfig=0
if [ -e /dev/shm/uml-felkonfigolva ];
 then
  echo "Már be van állítva a host oldali hálózati interfész."
  konfig=1
 else
  sudo tunctl -u 1000
  sudo chmod a+rw /dev/net/tun
  touch /dev/shm/uml-felkonfigolva
fi

gnome-terminal -e "/home/gergo/bme/virttech/linux' ubda='/home/gergo/bme/virttech/uml.img'
mem=512M eth0=tuntap,tap0'

if test $konfig -eq 0
then

echo -n "Várok a kernel indulására"

for ((  i = 0 ;  i <= 35;  i++  ))
do
  sleep 1
  echo -n "."
done
echo "."

sudo ifconfig tap0 192.168.0.254 up
sudo bash -c 'echo 1 > /proc/sys/net/ipv4/ip_forward'
sudo route add -host 192.168.0.222 dev tap0
sudo bash -c 'echo 1 > /proc/sys/net/ipv4/conf/tap0/proxy_arp'
sudo arp -Ds 192.168.0.222 eth0 pub

fi

echo "Kész."
sleep 3
```

uml-start.sh

```
#!/bin/bash
gnome-terminal -e '/home/gergo/bme/virttech/terminal.sh'
```

5. Gyakorlati felhasználás – grafika, további fájlrendszerek

Az UML rendszerem itt már egyre inkább használható állapotba került. De mi a helyzet akkor, ha elfogy a hely a root partícióról? Vagy csak egyszerűen szeretném elérni a host fájlrendszerét? Esetleg ha grafikus felületre van szükségem?

Az első kérdésre a válasz: az UML kernel parancssorán több blokkos eszköz is megadható, így szinte tetszőleges számú RAW lemezképet vagy fájlrendszerképet adhatunk hozzá a virtuális géphez.

```
~$ ./linux ubda=uml.img ubdb=uml2.img mem=512M eth0=tuntap,tap0
```

A host fájlrendszerének elérésére több lehetséges módszer is van. Kézenfekvő, hogy ha már úgyis van egy működő hálózati kapcsolatunk, NFS-el exportálhatjuk a host root-ját. Ezt az ötletet elvetettem. Egyrészt, mert nem fordítottam bele a kernelbe az NFS támogatást, másrészt meg mert erre a célra létezik az UML-nek beépített eszköze. Ez pedig a hostfs. A használatához mindössze egy mount parancs kiadására van szükség az UML-en belül.

```
~$ mount none /mnt/host -t hostfs
```

Innentől az UML-ben elérhető a host gyökérfájlrendszere a /mnt/host könyvtár alatt. A hozzáférési jogosultságokat természetesen lehet korlátozni és a gyökeret is meg lehet változtatni. Erre egy példa, ha a /hostfs könyvtár tartalmát szeretnénk elérhetővé tenni és benne lévő fájlok megnyitását csak hozzáfűzésre engedélyezni:

```
~$ ./linux ubda=uml.img mem=512M eth0=tuntap,tap0 hostfs=/hostfs,append
```

A hostfs egyébként root fájlrendszerként is használható, de ezért vagy funkcionalitást (a fájlok jogosultságainak megváltoztatása) vagy biztonságot (az UML kernel root-ként való futtatása) kell áldozni.

A grafikus felületre szintén két megoldást próbáltam ki. Az egyik egy kereskedelmi szoftver, a NoMachine NX Server terméke, a másik pedig az OpenSSH X forwarding funkciója. Mindkét megoldáshoz szükség van a virtuális gépben egy működő OpenSSH szerverre és egy X kiszolgálóra.
Az NX Server-t egész egyszerűen le kell tölteni .deb csomagként a cég weblapjáról, és dpkg-vel telepíteni. A host gépen aztán a hozzátartozó klienset futtatva egy teljes munkamenet vagy akár csak egyetlen program is elindítható vele.

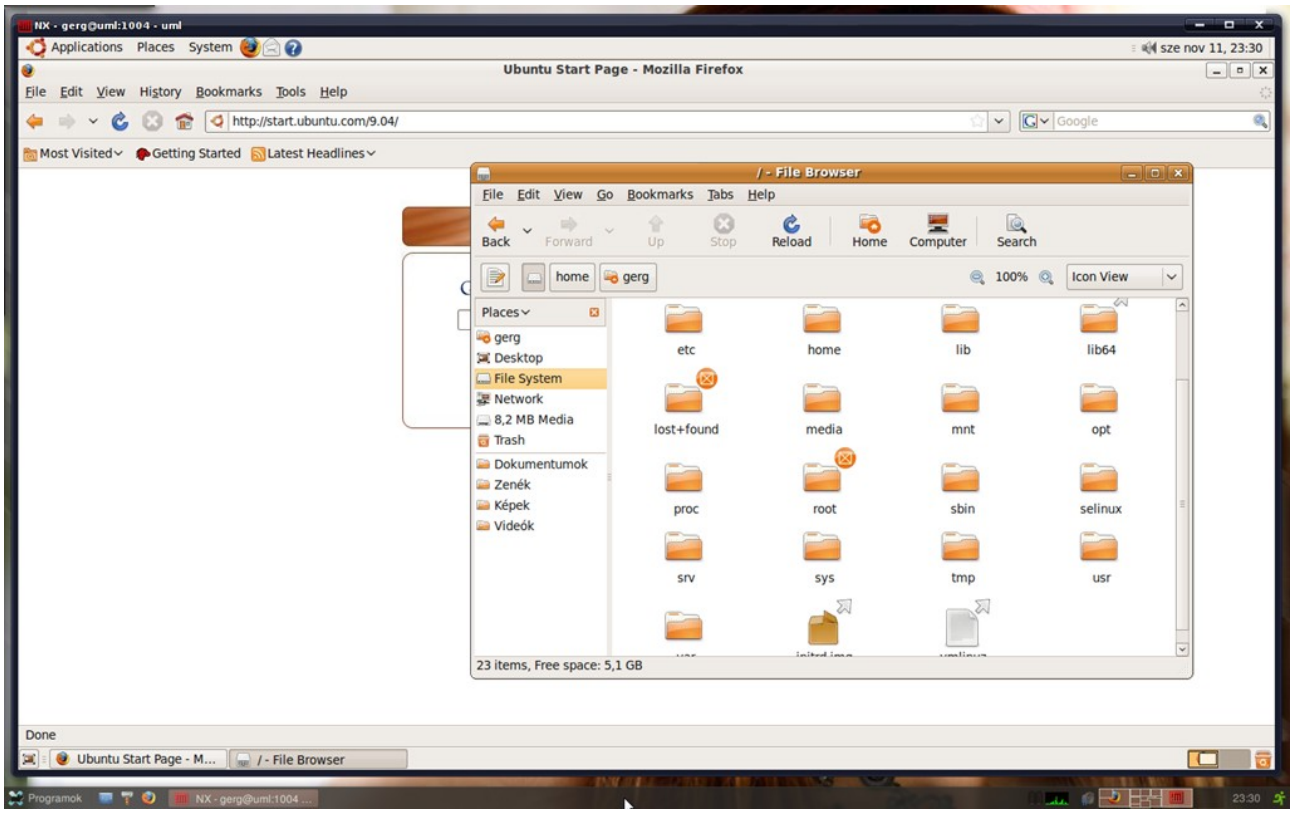
A kettes ábrán egy az NX klienssel indított gnome-munkamenet szerepel, a hármason ugyanezzel a klienssel futtatva egy önálló firefox példány. Itt látszik, hogy a GTK2 beállításaira és témájára vonatkozó környezeti változók hiányában az ablak a default értékeket veszi fel a host rendszeren.
Mindkét ábrán látható, hogy a vendég rendszerről származó ablakok megjelenése – szemben az X forwarding megoldással – eltér a hostétól.

A negyedik ábra egy az OpenSSH X forwarding képességét használó, szintén az UML-en belül futó firefox példányt mutat. Ezt a megoldást leginkább a kereskedelmi platformvirtualizációs termékek asztalba integráló módjához tudnám hasonlítani. Ám amivel ez többet tud: a guest rendszerből származó ablak nem csupán együtt lesz látható a többi ablakkal, a GTK, GTK2 és Qt ablakok host rendszeren érvényes tulajdonságait is felveszi, sőt, még a compiz által generált effektek is megjelennek rajta, így szinte észrevehetetlen, hogy valójában egy virtuálisan futtatott alkalmazással van dolgunk.

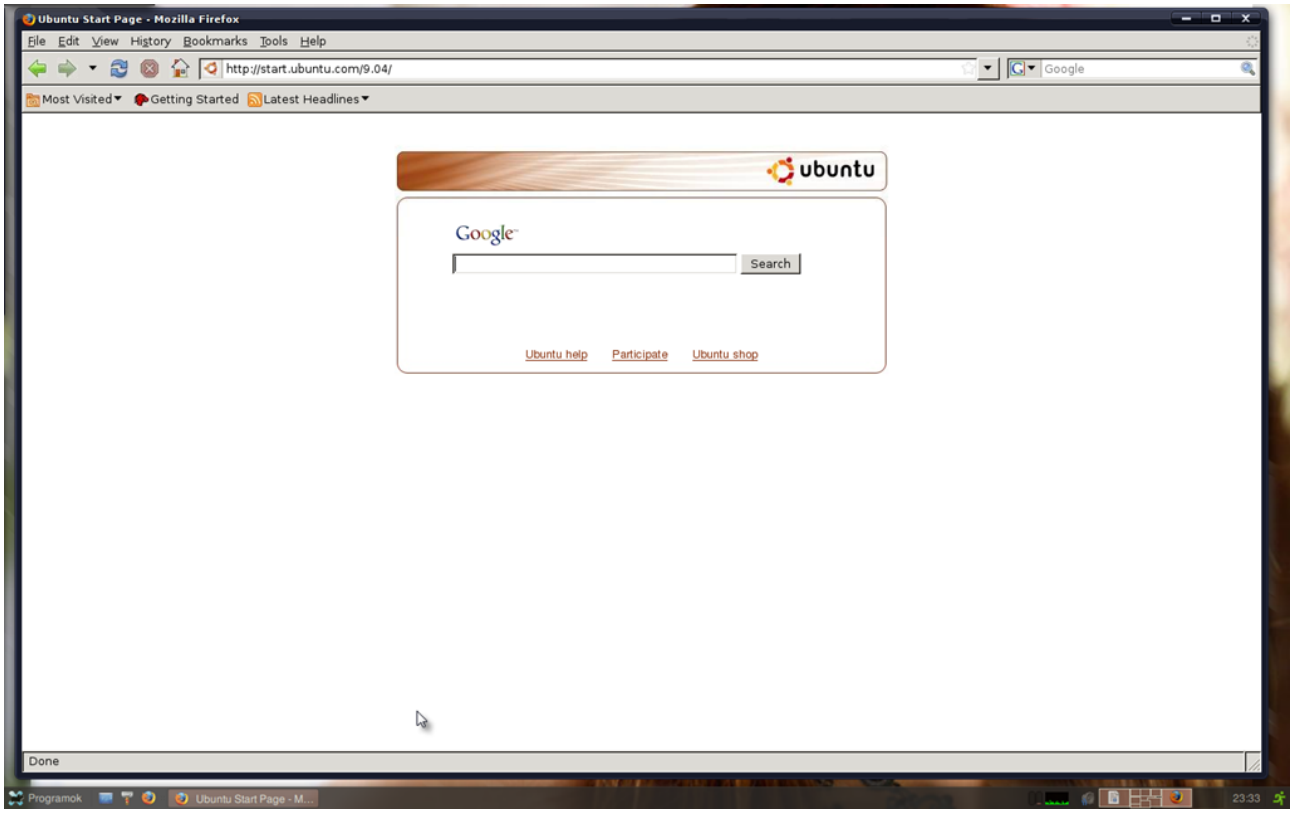
Sebesség szempontjából egyébként a Nomachine NX client érezhetően jobban teljesít, mint az SSH X forwardingja, akár hosszabb távon, munkára is alkalmas megoldás lenne.

Létezik egy direkt az UML-hez írt X változat is, mely képes megjeleníteni a host asztalán az UML-ben előállított képet, de ezt túl bonyolult lett volna beállítani, így ennek kipróbálásától eltekintettem.

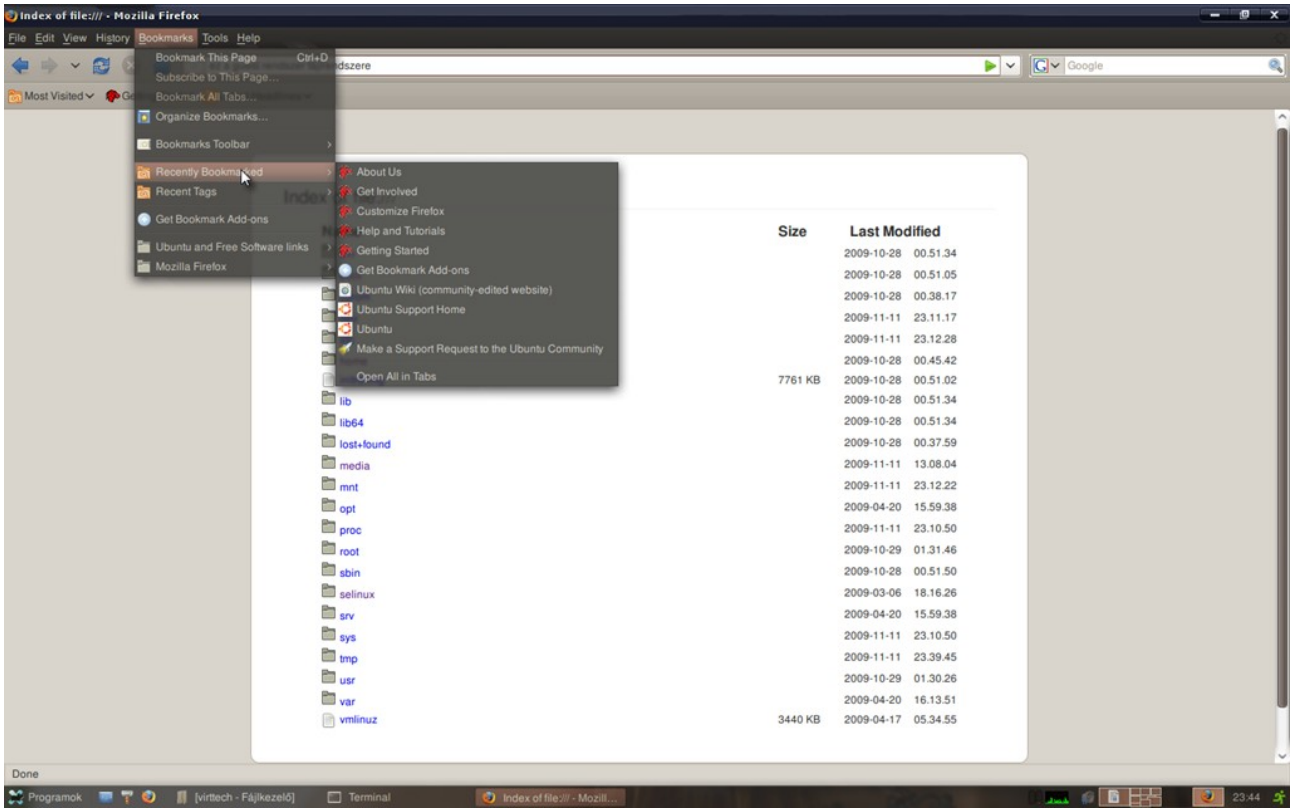
Az ábrák:



2. ábra: Grafikus felület NX klienssel teljes munkamenet módban



3. ábra: Grafikus felület NX kliensben, egyetlen program futtatásakor



4. ábra: Grafikus felület az OpenSSH X forwarding használatával

6. Szüneteltetés és az emulált eszközök módosítása futás közben, COW

Az UML-hez létezik egy uml_mconsole nevű eszköz. Azonban ahhoz, hogy ezt használni tudjuk, egy egyedi azonosítót kell rendelnünk az irányítani kívánt UML példányhoz:

```
~$ ./linux ubda=uml.img mem=512M eth0=tuntap,tap0 umid=ubuntu
```

A konzolhoz ezzel a paranccsal lehet hozzáférni:

Ekkor kapunk egy szöveges konzolt, melyen keresztül lehetőségünk van különböző utasításokat

```
~$ uml_mconsole ubuntu
```

adni az adott UML példánynak. Így átmenetileg felfüggeszthetjük a futását, leállíthatjuk, kiírathatjuk a kernel verziószámát, hozzáadhatunk hálózati kártyákat vagy blokkos eszközöket a virtuális géphez, a nélkül, hogy le kellene állítanunk azt. A lehetséges parancsok listája megtalálható a project weboldalán.

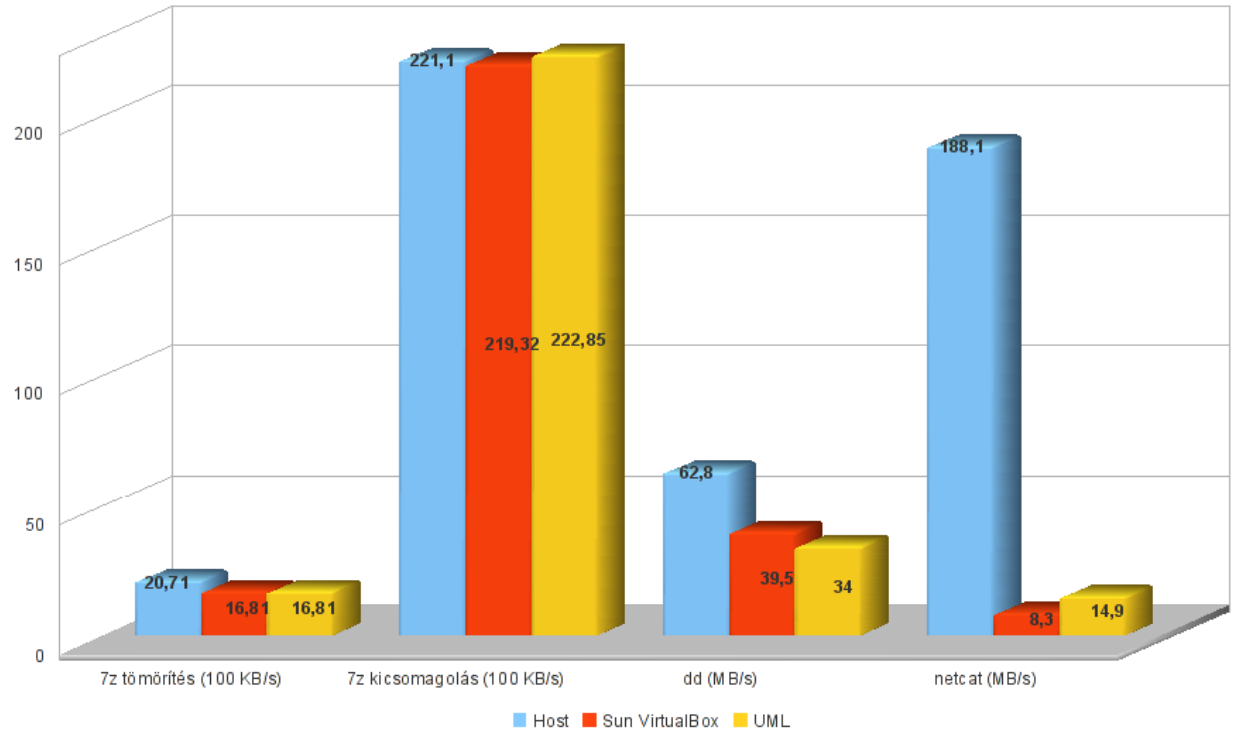
Érdekes lehetőség még, hogy az UML-en belül futó folyamatok számára biztosított egy interfész, melyen keresztül üzeneteket tudnak megjeleníteni ezen a konzolon.

Az UML beépített blokkos eszköz meghajtója képes COW képek kezelésére is. Ez azt jelenti, hogy minden blokkos eszközhöz megadhatunk a parancssoron egy másodlagos fájlt. Ekkor az elsődleges fájl csak olvasható lesz, a módosítások a másodlagos fájlba kerülnek kiírásra. Amennyiben ez a fájl egy sparse fájl, az általa elfoglalt lemezterület kezdetben mindössze néhány KB, majd a mérete szükség szerint nő.

```
~$ ./linux ubda=uml.img,cowfile.img mem=512M
```

7. Teljesítmény

A virtuális gép teljesítményének vizsgálatára összesen három, elég szintetikusnak mondható tesztet végeztem. Egyrészt futtattam egy 7z benchmarkot, másrészt a virtuális merevlemezről a /dev/null eszközre másoltam adatokat (fizikailag a lemezképek ugyanazon az eszközön voltak), harmadrészt netcattel teszteltem a virtuális hálózat áteresztőképességét. Az összehasonlítási alap a host gép teljesítménye volt. Az eredmények:



5. ábra: A teljesítményteszt eredményei

A méréseket többször megismételtem, és a kapott eredményeket átlagoltam, a dd tesztnél külön figyelve arra, hogy a végeredményben a disk cache hatása ne jelentkezzen.
Az eredmények nagyjából megfelelnek annak, amit vártam, kivéve, hogy a 7zip benchmark a kicsomagolási tesztben az UML kernel felett minden alkalommal jobb teljesítményt mért, mint ha közvetlenül a host kernel felett futtattam. Hogy ez miért van így, azt nem sikerült megfejtenem.

7+1. coLinux

Elvileg a Cooperative Linux hasonló a User-mode Linuxhoz, csak éppen windows host felett fut. Ennek kipróbálására csak nagyon kevés időt szántam, és miután már az alkalmas fájlrendszer elkészítésénél problémákba ütköztem, inkább nem foglalkoztam vele tovább.