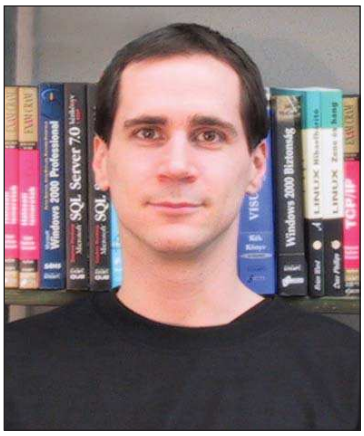
Nézzük, mivel bosszanthatjuk fel a hunglizmus híveit e hónapban – legalábbis sok olvasó teljesen biztos benne, hogy valami ehhez hasonló mottó jegyében írom meg minden hónapban a szótároldalt. Természetesen nem az a célom, hogy csökkentsem a hipotóniás esetek számát, és szerencsére a legtöbb szakember egyetért a magyarításokkal. Amihez – sajnos – kezdek hozzászokni, a fiatalok szemében látható csillogás, mikor szinte kéjesen mesélik, mennyire vagány, mások által érthetetlen tolvajnyelvet alakítottak ki baráti körükben (melyről később rendszerint kiderül, hogy csupán az irc és az ad&d angol nyelvezetének egy fura, minden nyelvtani szerkezetet nélkülöző, eltorzult változata). Szerencsére azonban idővel az ember rájön, hogy a nyelv milyen fontos – nem csupán a pontos és érthető beszéd, de az írott nyelv, sőt a gondolkodásra gyakorolt hatása is. És szinte természetes, hogy a nyelv fejlődése során több szó „próbálja ki magát” egy-egy fogalom képviseletében.
Talán pont ez bosszantja leginkább az informatikusokat. Micsoda feszültséget tud szülni, hogy (veszőparipámra utalva) a meghatározás és a definíció ugyanazt jelentik! Vagy hogy az ember külföldön megtanulja, hogy process, és idehaza folyamatnak nevezik! Ezek egyszerű példák, mindenkinél máshol van a határvonal. Van, akinek természetes, hogy útválasztókról beszél, van, aki még a rendszergazda helyett is administratort képes mondani. Vannak olyan szavak, kifejezések, amelyek a „mesgyén” vannak, egyszerűen nem tudjuk eldönteni, hogy melyik szót használjuk az újságban. Ilyen például az abszolút útvonal, ami helyett sokszor a kifejtett vagy teljes, teljesértékű útvonalat használják, vagy ott van az I/O rövidítés, ami mindenki számára a be- és kimenetet jelenti, sőt elsőre komoly kihívás az I/O magyar megfelelőjének, a B/K-nak a megfejtése.
Vannak olyan szavak is, amelyekre nemigen találunk a szakmában „hagyományos” fordítást. Ebben a hónapban ismét előkerült a carrier grade, ami a „távközlésben használatos” értelemmel bír, mégis nehéz lefordítani, főleg, amikor valamelyik Linux-változat „Carrier Grade Edition” termékéről beszélünk. Sokkal nagyobb gondban voltunk a stackable filesystem kifejezéssel, ami lényegében olyan fájlrendszert takar, amelyik képes „ráülni” egy másik fájlrendszerre. Azért is nehéz kérdés, mert a stack mint adatszerkezet is szerepel, valamint a stackable mint fogalom is (például a stackable router jelentése tornyozható vagy halmozható útválasztó – amikor több útválasztót össze tudunk kötni, és azok egyetlen egységként képesek dolgozni). A végén – talán nem a legszerencsésebben – a halmozható fájlrendszer mellett maradtunk, mondhatnók, a pocakcsapkodás módszerét alkalmazva.
Ezúton is kérek mindenkit, ha ötlete van az újságban lévő szavak magyarításával kapcsolatban, ossza azt meg velünk!
Jó olvasást!

---

access point → hozzáférési pont
adatfolyam – data stream
adatsín – data bus
alulcsordulás – underrun
array → fürt
bővítmény – plugin
brute force – nyers erő
burkoló – wrapper
bus – sín
carrier grade – távközlésben használatos, nagy adatátvitelhez kialakított, távközlési
chipset – lapkakészlet
cluster → telep
code-freeze → kódfagyasztás
codec → kodek
data bus – adatsín
data stream – adatfolyam
decoder – visszafejtő
encoder – kódoló
érintőképernyő – touch screen
event → esemény
farm → telep
figyelő – listener
filter → szűrő
folt – patch
folyam – stream
force feedback – (jó ötleteket várunk!)
fürt – (array) valamilyen szempontból azonos típusú eszközök együttese, amelyet logikailag egy eszközként kezelhetünk. Amennyiben teljes értékű gépeket állítunk össze, telepekről beszélünk.
halmozható fájlrendszer – (stackable filesystem) más fájlrendszerekre ráépülni képes, további szolgáltatásokat biztosító kiegészítés.
hard link → közvetlen hivatkozás
hozzáférési pont – access point
hurokhívás – (recursion) az önhívás egy változata, amikor közvetve történik meg az önhívás (például két függvénynyel kialakított önhívás).
illegal operation exception → érvénytelen műveleti kivétel
init →rendszerindítás és -beállítás
inode →fájlleíró
interprocess communication – folyamatközi (folyamatok közti) kapcsolattartás
Interrupt Descriptor Table → IDT, azaz megszakításleíró tábla
joystick → botkormány
kapu → port
kernel panic → magpánik
kodek – (codec) kódolást és visszafejtést is végző rész, ami például lejátszókban mint kiegészítő modul használható.
kódoló – encoder
közvetett hivatkozás – soft, symbolic link
közvetlen hivatkozás – hard link, másodbejegyzés
lapkakészlet – chipset
listener → figyelő
nyers erő – (brute force) A nyers erő módszerének hívjuk a különleges vagy ügyes megoldások nélküli, egyszerű módszereket, amelyek (más megoldásokhoz képest) komoly erőforrást igényelnek.
önhívás – recursion
patch → folt
plugin → bővítmény
port – kapu
privileged instructions → a program jogosulatlanul szeretne hozzáférni a magból a tárterülethez, vagy hibásan hajt végre kiváltságokhoz kötött utasításokat.
recursion – önhívás, hurokhívás
segmentation – szakaszolás
segmentation fault → szakaszolási hiba
sín – bus
soft link → közvetett hivatkozás
symbolic link → közvetett hivatkozás
stackable filesystem → halmozható fájlrendszer
stackable device → tornyozható eszköz
structure→ szerkezet
szakaszolás – segmentation
szűrő – filter
telep – farm, cluster
touch screen → érintőképernyő
underrun → alulcsordulás
visszafejtő – decoder
wheel → görgő (egérgörgő)
wrapper → burkoló

Beköszöntő



Szy György
a Linuxvilág főszerkesztője,
a Kiskapu Kiadó vezetője.
Mindenki levelét örömmel
várja a következő levélcímen:
Szy.Gyorgy@linuxvilag.hu

Móka és kacagás

Sokan panaszkodtak, hogy keveset foglalkozunk a játékokkal, sőt egy rendszer nem is igazi rendszer, ha nem lehet rajta jóízűt játszani. Hiszen már az iskolában is azt tanítják, hogy könnyedén, vidáman kell venni az életben elénk kerülő akadályokat. Régebben egy-egy játék erejéig kitekintgettünk a „száraz" szakmai világból, de most külön csemegéket nyújtunk át játékos kedvű olvasóinknak! A 74. oldaltól kezdődően ugyanis egy egész kis mellékletnek beillő rész foglalkozik a linuxos játékokkal. Sőt még Marcel is kiruccan a fogadó nyugodt világából egy kis könnyed fénymotorozásra (68. oldal). Szerencsés egybeesés, hogy a Gentoo is most erősített be a linuxos játékok területén.

De nem csupán a kikapcsolódás oltárán feláldozott idő eltöltésének minőségi színvonalával foglalkozunk e hónapban. Ahogy már megszokhattuk, találkozunk több rendszergazda-nevelő cikkel is. Nagy örömömre méltó testvért találtunk ***dr. Büki András*** cikke mellé (Héj-programozás, 2. rész, 59. oldal), ami a szabályos kifejezéseket mutatja be (40. oldal). No igen, a már klasszikus idézet a klasszikustól: /(bb|[^b]{2})/.

A cikkek között a Vezérfonal e hónapban a rendszermag, ezen a területen is sok érdekeset igyekeztünk belekanalazni ebbe a nyolcvanegynéhány oldalba. A 21. oldalon kezdődő cikkében ***Robert Love*** igyekszik nekünk röviden bemutatni, hogy milyen újdonságokat várhatunk a 2.6-os sorozattól, de foglalkozunk még a titkosító API-val (32. oldal), a magmódú programokkal (29. oldal) és a 2.5-ös beállításával (26. oldal) is. A témát pedig egy érdekes kitekintés zárja, aminek révén a fájlrendszerek világába kukkantunk be.

Ez utóbbi témával kapcsolatban ismét egy nagyon nehéz magyarítással kapcsolatos kérdésbe futottunk, ez pedig a stackable filesystem, de erről részletesebben az előző oldalon lévő szótároldalon írok.

Mindig örömömre szolgál, ha valamilyen ügyes megoldást mutathatunk be. E hónapban is sikerült ilyeneket közzétenni. Az első, ami igazából egy külön „világ", a mikrovezérlők programozásával kapcsolatos, rejtelmeibe ***Havránek Ferenc*** vezet be bennünket (37. oldal). De hasznos megoldást mutat be – igaz, egy teljesen más területről – ***Fülöp Balázs*** is a 64. oldalon. És egyből utána következik ***Kolcza Péter*** cikke a házi multimédia könnyed megvalósításának lehetőségeiről.

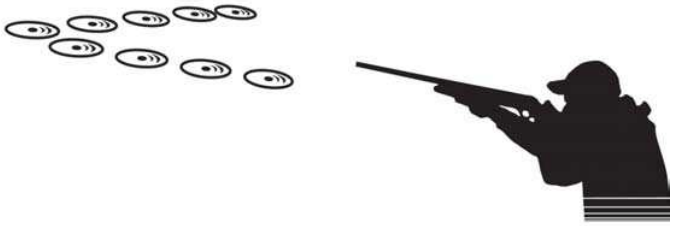
Vállalatirányítási rendszerek

A több hónapja rágcsált téma nem halt el, az ötleteket továbbra is gyűjtjük!

E hónapban a GKM és az IHM által kiírt két pályázatról szóltam. Érdekes összehasonlítani, hogy a két különböző szempont szerint működő minisztérium mennyire másként ír ki egy ilyen pályázatot. És főleg, hogy mennyire másként állnak hozzá a nyílt alapú rendszerekhez. Sajnálattal vettem tudomásul, hogy az IHM pályázati kiírásából szinte üvölt, hogy a pályázó nyílt rendszerekben nem is gondolkodhat.

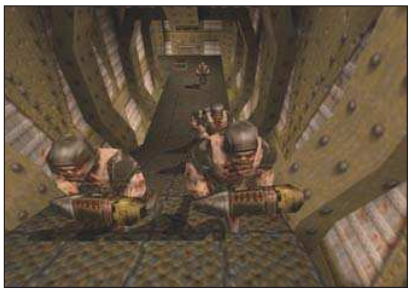
Talán abból is egyértelműen következik ez, hogy a megpályázható rendszer mögött nem elég, hogy cégnek kell állnia, de a cég tavalyi (rendszereladásokból befolyt) bevétele meg kell haladja a százmillió forintot. Nos, ha jól tudom, pont annyi ilyen cég van jelen a hazai piacon, ahány ezeréves tölgy az Astorián. Csendben reménykedem csupán, hogy jövőre több tucat olyan megoldásszállító legyen szent hazánkban, akik nyílt rendszerekből ekkora bevételt könyvelhettek el.

Programvadászat



Játékok

E számunktól kezdve megpróbáljuk a játékokat kedvelő Linux-rajongóknak megmutatni, hogy mit is lehet kedvenc operációs rendszerünk alatt játszani.



A korongon helyet kapott a Quake játék mindhárom változata, amelyekhez részletes telepítési útmutatót kaphatnak játék rovatunkból.
Másik két játékos cikkünkhöz olvasóink szintén a korongon találhatják meg a programokat, ezek telepítéséhez az archívfájlokban találhatunk útmutatót.

Opera

Az Opera böngésző fejlesztőinek révén kirepültek a 7. sorozat első fecskéi. Ezek csak a kipróbálásra szánt változatok, úgyhogy senki se lepődjön meg azon, ha valami furcsaság történik a használata közben. Én ilyet nem tapasztaltam, pedig nyüvöm egy ideje. Még mindig hihetetlenül kis helyet foglal mind a

merevlemezen, mind pedig a memóriában – ennek köszönhetően gyorsan elindul még a régebbi gépeken is. CD-mellékletünkön megtalálhatják a legfrissebb 7.11 beta2 változatot, többféle formátumban is. Szerepel közöttük rpm, deb, tar.gz, tar.bz2 és ppc.rpm fájl, különféle fordítókkal fordítva (gcc-2.95, gcc-2.96 és gcc-3.2). A telepítése nagyon egyszerű, az rpm alapú rendszereken a megfelelő könyvtárban adjuk ki az `rpm -i opera_csomag_neve.rpm` parancsot, deb csomagnál ez `dpkg -i opera_csomag_neve.deb`, ekkor a telepítőrendszer megvizsgálja, hogy nekünk tényleg erre a csomagra van-e szükségünk. Amennyiben igen, akkor egy Opera böngésző boldog tulajdonosai és használói lettünk. Kicsit bonyolultabb a telepítés a tar.gz fájlokból: ezeket a fájlokat ki kell csomagolnunk, majd az `install.sh` fájlt kell a kicsomagolt könyvtárban futtatnunk.
A korongra feltettük a megbízható 6.12-es Opera-változatot is azok számára, akik nem szeretik a próbaváltozatokkal érkező esetleges hibákat.

Mozilla

1.3

A legfrissebb megbízható Mozilla az 1.3.1, ez mind hálózati telepítő, mind teljes telepítőcsomagban felkerült a korongra.

1.4

Az 1.4-es fejlesztői Mozilla-változat azoknak kedvez, akik szeretik kipróbálni az újdonságokat. Ennél a programnál is meg kell említenem, mint ahogy tettem ezt az Opera böngésző próbaváltozatánál is, hogy csak azok használják, akik tudják, mire is vállalkoznak ezzel a

lépésükkel. Itt is előfordulhat hibás, rendellenes működés, engem azonban az új dolgok hatalmas csábító erejükkel mindig meggyőznek arról, hogy tudom, mit csinálok! Így tettem a Mozillával is az Opera mellett, mindig merevlemezemre kerül a legfrissebb változat is. Telepítése mindkét esetben rendkívül egyszerű: csomagoljuk ki az archívumokat, lépjünk be a keletkezett könyvtárba. Rendszergazdaként (root) futtassuk a `mozilla-installer` parancsot, ha mindenki számára elérhetővé akarjuk tenni a programot, vagy egyszerű felhasználóként, ha csak mi akarjuk használni, esetleg nem rendelkezünk megfelelő jogosultságokkal a rendszeren. Ha felhasználóként telepítjük, meg kell változtatni a telepítési célkönyvtárat.

SuSE-frissítések

Az előző számunk mellékletén megjelent SuSE-kiadáshoz adjuk most közre a frissítéseket. Sok olvasónk kérdezte, hogy honnan érhető el a cikkemben leírt FTP-archívum, ahonnan a teljes csomaglistát elérhetik a 8.2-es változathoz.
A helyzet az, hogy időben annyira elémentünk a dobozos változat megjelenésének, hogy a magazin megjelenésekor az FTP-kiszolgálókra még nem kerültek ki a csomagok. Most már elérhető bárki számára, íme a kiszolgálólista:
➲ ftp://gd.tuwien.ac.at/linux/suse/suse.com/i386/8.2/
➲ ftp://ftp.solnet.ch/mirror/SuSE/i386/8.2/
Sajnos magyar tükröt eddig nem találtam (bevallom, nem is nagyon kerestem, csak néhányat néztem meg), nincs fent az anyag az ftp://ftp.suselinux.hu-n sem, csak egy fájl, ami HAMAROSAN… névre hallgat.

Rendszermag

E havi lapszámunk fő témája a Linux-rendszermag, ezekhez a cikkekhez kapcsolódik szervesen a rendszermag-könyvtárban a legfrissebb fejlesztői mag, ami a 2.5.69-es változatszámú. Ezen már mindenki bátran kísérletezhet, és ismerkedhet a (reméljük!) hamarosan megjelenő 2.6-os vagy 3-as megbízható rendszermag újdonságaival, változásaival. Szükség is van az ismerkedésre, mivel a fejlesztők rendesen átalakították néhány helyen, s ez a beállításra is vonatkozik.
Jó kísérletezést!

Csontos Gyula
(Csontos.Gyula@linuxvilag.hu)
A Linuxvilág szakmai és CD-szerkesztője. Szabadidejében szívesen mászik hegyet és kerékpározik.

Juhu!

Közel két év kitartó munkájának eredményeképpen megszületett az UHU-Linux 1.0 Office. Az elsősorban irodai és otthoni felhasználásra felkészített UHU-Linux az eddig nyilvánossá tett kiadás előtti változatokhoz képest megújult külsővel és minden eddiginél frissebb tartalommal kerül terjesztésre. Hazánk nemzeti Linux-terjesztése élen jár a magyar nyelv támogatásában: minden idők messze legjobb magyar helyesírás-ellenőrzőjével – ami az OpenOffice.org-ban megtalálható myspell-motor továbbfejlesztett változata –, a többi linuxos terjesztéshez képest rendkívül jó Gnome- és KDE-fordításokkal.
Kimondottan a magyar felhasználók igényeinek kielégítésére, előre telepített változatban megtalálható a MKOGY CompLEX CD Jogtár, valamint a Lafisoft raktárkezelő és számlázóprogram.
Az UHU-Linux 1.0 irodai összeállítására méltán büszkék lehetünk – gratulálunk a fejlesztőknek! A változat letölthető vagy kereskedelmi termékként ajándékokkal, könyvvel megvásárolható az UHU-Linux honlapján.
A terjesztés várhatóan a számítástechnikával foglalkozó üzletekre, könyvesboltokra és áruházakra egyaránt kiterjed.
➲ http://www.uhulinux.hu

Gibizer Tibor

United Devices: Grid MP 4.0

A United Devices, az osztott hálózati alkalmazások piacvezető gyártója bejelentette Grid MP 4.0 termékét. Az új változat teljesen új szemléletet alkalmaz az erőforrások, a felhasználók és a házirendek virtuális kezelésében, üzenettovábbító felületének köszönhetően pedig a korábbinál jóval több alkalmazást lehet – akár módosítás nélkül – lazán összekapcsolt csomópontokból álló hálózaton futtatni.
Szintén újdonság a Linux alapú alkalmazások windowsos futtatásának lehetősége, aminek révén a felsőbb kategóriás gépekről kilépve jelentős, korábban vélhetően kihasználatlan számítási teljesítményt lehet az adott vállalat szolgálatába állítani.
A United Devices megoldása adja a grid.org szervezet hátterét is. Aki rák-, antrax- vagy himlőkutatási célokra szeretné felajánlani számítógépének felesleges processzoridejét, az ügyfélprogramot a ➲ http://www.grid.org oldalról töltheti le.
➲ http://www.ud.com

Túlcsordulás-védelem rendszermagszinten

Ingo Molnár fontos linuxos biztonsági bővítést jelentett be május elején. Az Exec Shield rendszermagfolt feladata az, hogy megvédje a rendszert a verem, a pufferek és a függvénymutatók túlcsordulásainak káros következményeitől – amelyek az „egyszerű” adatmódosulástól egészen a rendszer feletti ellenőrzés támadó általi átvételéig terjedhetnek. A hatékony, a rendszer számára minimális többletterhelést jelentő folt működése a felhasználók és az alkalmazások számára átlátszó, a programok újrafordítására nincs szükség. A folt bizonyos típusú hibák következményeinek elhárítására ugyan nem képes, ám a túlcsordulások miatti gondok túlnyomó részét várhatóan elfeledhetjük.

P-sorozat a Matroxtól

Két új grafikus kártyát mutatott be a Matrox, P650 és P750 jelzéssel, egyben – a hosszú éveken át futott G-sorozat után – életre hívva a P-sorozatot. A játékosokról már szó sincs a cég közleményében, hiszen – sokak bánatára – a Matrox termékei teljesítményüket tekintve jó

ideje elmaradnak az ATi és az nVidia lapkáira épülő kártyák mögött. A teljesítmény azonban nem minden; legendásan jó képminőségének köszönhetően a Matrox biztos piacra lel a grafikusok, tervezők körében, akik két-három kijelzővel felszerelkezve a korábbinál jóval hatékonyabban tudják végezni munkájukat.
Mindkét kártya Parhelia-LX lapka köré épül, 8× AGP felületre csatlakozik és 64 MB DDR RAM-ot tartalmaz. A P650 két megjelenítő vezérlésére képes – ezek hagyományos monitorok, analóg és digitális TFT-panelek egyaránt lehetnek, illetve a második kimenetre tévé is csatlakoztatható. A P750 ennél is többet tud, bizonyos korlátozások révén akár három kijelzővel is elboldogul. A P650 ára körülbelül 38 ezer forint lesz, a P750 pedig 54 ezer forint körül lesz megszerezhető. Lapzártánk idején mindkét kártya előrendelése lehetséges már a Matrox oldaláról.
➲ http://www.matrox.com

USB-s csecsebecse-rajongók, figyelem!

Újabb taggal bővült az SMC EZ Networking termékcsaládja: egy USB 2.0 felületre csatlakozó 10/100 Mb/s sebességű ethernetcsatolóval. Hasonló eszközöket eddig is lehetett vásárolni, ám inkább USB 1.1-es felülettel, ami a maga 12 Mb/s sebességével nem tette lehetővé az ethernet kínálta átviteli lehetőségek kihasználását – a szűk keresztmetszet az USB 2.0 alkalmazásával végre eltűnt.
Az SMC új csatolója célszerű kialakítású, apró bütyökként ékesíti használójának számítógépét, bár figyelembe véve, hogy ethernetcsatoló nélkül ma már elég nehéz számítógépet venni, alkalmazási köre elég szűknek tűnik. Akinek mégis jól jönne egy ilyen játékszer, 33 dollárért szerezheti meg.
➲ http://www.smc.com

Sikeres az iTunes

Az Apple Computer internetes zeneboltjának megnyitását erős kételkedés övezte. Sokan öltek már hatalmas összegeket egy-egy internetes szolgáltatásba, és nagyon sokan buktak vele hatalmasat. Nem így az Apple, amelynek új iTunes Music Store szolgáltatása elsöprő sikert aratott: nyitás után a kizárólag Apple-felhasználók számára elérhető oldalon körülbelül 275 000 zeneszámot adtak el, darabonként 99 centes, vagyis közel 230 forintos áron. A vásárlók letöltés után CD-re írhatják vagy iPod készülékre tölthetik át a zeneszámokat, az Apple tehát egyben saját hordozható lejátszója iránt is erősítheti a keresletet, mindent átfogó szolgáltatást nyújtva ügyfeleinek.
A szolgáltatás kiterjesztését év végére tervezik, ami ekkor várhatóan már PC-ről is elérhető lesz.
Az iTunes sikerét jelentős részben az magyarázza, hogy míg CD-n 16–18 dollár egy album, addig letöltve mindössze 10 dollárt kell fizetni érte. A lemezkiadók is nyilván jól járnak, és miután hosszú éveken harcoltak a zeneszámok internetes terjesztése ellen, végre hasznot húzhatnak a hálózati eladásokból. Amennyiben az Apple szolgáltatása hosszú távon is életképesnek bizonyul, illetve elérhetőségét kiterjesztik, végre megkezdődhet az a változás a zeneszámok kereskedelmében, amire hosszú ideje várunk.
➲ http://www.ipod.com/music/store/

Nosztalgiázó Verbatim

A Verbatim új „vinil" CD-lemezeire – amelyek a régi bakelitlemezek világát idézik – tintasugaras nyomtatóval is nyomtathatunk. Nosztalgialemezeket eddig is gyártott a Verbatim, ám a lemezek felületére – erre alkalmas felület, címke hiányában – eddig nem lehetett nyomtatni.

A 80 perces CD-ket, amelyek minden tekintetben a régi lemezeket idézik – még barázdákat is találunk a felületükön –, elsősorban azoknak ajánlják, akik régi felvételeiket korszerűbb, a lejátszások alkalmával nem sérülő adathordozóra szeretnék másolni. A nem csak zene tárolására alkalmas lemezek nyomtatható címkéjüknek köszönhetően egyedi megjelenéssel ruházhatók fel, és jól felhasználhatók céges anyagok, egyedi összeállítások, bemutatók terjesztésére is. A Verbatim normál lézerlemezeihez hasonlóan ellenállnak az UV-sugárzásnak, a karcolásoktól pedig kettős védőréteg óvja őket. Az élettartam-garanciával kapható lemezek ára – 50-es csomagban – 50 dollár.
➲ http://www.digitalvinylcdr.com

Ínyenceknek: PlexWriter Premium

A Plextorra nem jellemző, hogy naponta jelentetne meg újabb és újabb terméket, ám az ilyen ritka pillanatok annál több érdekességet tartogatnak. Nincs ez másképp a legújabb PlexWriter Premium CD-íróval sem, ami 52×-es írásra, 32×-es újraírásra és 52×-es olvasásra képes. A meghajtóhoz mellékelt PlexTools segédprogrammal különleges, a Plextor meghajtóját egyedivé varázsoló lehetőségek egész sorát aknázhatjuk ki.

- SecuRec: a lemezek jelszóval védhetők.
- GigaRec: egy 700 MB-os lemezre 1 GB anyagot lehet rögzíteni.
- Q-Check: írás után ellenőrzi a lemez minőségét és esetleges hibáit.
- Silent Mode: a tálca mozgatásának, a lemez felpörgetésének és olvasásának sebessége – és ezzel zajossága is – szabályozható.
- VariRec: a lézersugár energiája állítható, így a zenei felvételek is kiváló minőségben rögzíthetők.

A meghajtó ATAPI-felületre illeszkedik, a gyorstár-alulcsordulások ellen 8 MB memóriával védekezik. Ára 30 ezer forint körül alakul majd.
➲ http://www.plextor.com

Linuxos kiszolgálók Opteron processzorokkal

Az AMD új, 64 bitesprocesszoraitól hangos a fél szakmai világ, a különféle cégek egymás sarkát taposva jelentik be, hogy támogatják az Opteron lapkák használatát. A kifejezetten Linux alapú megoldásokat szállító Penguin Computing sem marad ki a sorból, Altus 1000E kiszolgálóival az AMD legújabb termékének sikeréből próbálja kihasítani a maga hasznát. Az állványba szerelhető gépeket kifejezetten fürtökben való használatra kínálják, erre utalnak az összeállítás adatai is. Az 1 egység magas gépházba kettő 200-as sorozatú Opteron processzor és legfeljebb 16 GB DDR RAM kerülhet. A gépeket lemezek nélkül is lehet rendelni, kettős gigabit ethernet-csatolóval bírnak, esetleges bővítésüket pedig PCI-X foglalat segíti. A cég hároméves garanciával kínálja kiszolgálóit, amelyekre előtelepítik a SuSE Enterprise Server 8 64-bites változatát.
➲ http://www.penguincomputing.com

Játékra fel!

A Lycoris és a TransGaming összefogásának eredményeként a játékosoknak többé nem kell ráfanyalodniuk a Windowsra – állítja a két cég. A közösen megjelentetett GamePak egyrészt öt nyílt forrású, kifejezetten a Desktop/LX operációs rendszerhez igazított, új hangokkal és grafikával feljavított játékot, másrészt egy hónapnyi WineX-előfizetést foglal magában.

A szintén most megjelent WineX 3.0 révén több mint 250 windowsos játék futtatható Linux alatt is, köztük olyan sikeres programok is, mint a Battlefield 1942, Medal of Honor vagy a SimCity 4 – természetesen ezeket már külön kell megvásárolni.
➲ http://www.lycoris.com
➲ http://www.transgaming.com

Újabb linuxos PDA

Mindentudó linuxos zsebtitkárt mutatott be a japán PalmNet. A készüléket állítólag cégeknek fejlesztették, amit további szolgáltatásokkal kiegészítve és testreszabva majd a saját nevük alatt dobhatnak piacra. A tenyérnyi mindenes nemcsak elektronikus noteszként, de mobiltelefonként és digitális kameraként is használható, illetve vezeték nélküli hálózati kapcsolatot is képes teremteni. Magát az eszközt a koreai IMRI fejlesztette, operációs rendszerét pedig a Pekingi Műszaki Egyetemen állították össze. Hasonló játékszert a Sony is kínál, ám kisebb tudással és drágábban. A körülbelül 560 dolláros árú készülék operációs rendszerét egy nonprofit szervezet segítségével nyilvánosan is elérhetővé fogják tenni.

Újabb IP-telefonok a Cisco kínálatában

A Cisco Systems új IP-telefonokat – szám szerint hármat – jelentett be. A Cisco 7920-as jelzésű mobil telefonkészülék 802.11b szabvány szerinti vezeték nélküli hozzáférési pontokon keresztül továbbítja a beszélgetéseket, így elsősorban olyan helyeken tehet jó szolgálatot, ahol a munkatársak egymástól távol, szétszórtan végzik a munkájukat.
A 7902G és 7912G készülékek asztali használatra készültek. Szabványos ethernethálózathoz csatlakoznak, és minden olyan szolgáltatást biztosítani tudnak, amit egy általános irodai telefontól a használója elvár: hívásismétlés, hívásátadás, hívásvárakoztatás, konferenciahívás stb. Mindkét készülék egy telefonszámot kezel, beállításaik TFTP-protokollon keresztül frissíthetők, valamint a DHCP-protokollt is támogatják. A Cisco ugyanakkor további telefonos és egyéb terméket, bővítést, alkalmazást is bejelentett.
➲ http://www.cisco.com

Windows helyett CrossOver

A CodeWeavers lassan új nevet is kereshet a 2.0-sá érett CrossOver Office-nak. A Linux–Microsoft-barátságot – több fontos windowsos alkalmazásnak Linux alatt is futtathatóvá való tételével – szorgalmasan ápoló program ugyanis a Microsoft Office XP összetevői mellett (az Outlookot kivéve) már az Adobe Photoshop 7-es változatát és az Accesst is támogatja. Az új változat immár a japán, kínai és koreai nyelvű alkalmazásokat is futtatja, újabb hatalmas piacokat nyitva meg a CodeWeavers előtt. Az új változat az összes nagyobb terjesztést támogatja, ára 55 dollár.

➲ http://www.codeweavers.com

IBM Linux-központ nyílt Londonban

Az IBM új, elsősorban a pénzügyi szakma igényeit kielégíteni hivatott Linux-tanácsadó központot nyitott Londonban. A központ feladata az, hogy a gyakorlatban is kipróbálhatóvá tegye azokat a Linux alapú megoldásokat, amelyeket az IBM kínál ügyfeleinek, és ilyen módon a Linuxra való áttérésre buzdítsa a bankokat, pénzügyi intézményeket. A központban különféle kiszolgálók és Linuxot futtató asztali gépek is találhatók, így akár továbbképzések tartására is alkalmas.

SuSE Linux a távközlésnek

Megjelent a SuSE Linux Enterprise Server Carrier-Grade Linux (CGL) Edition változata. A UnitedLinux alapokon, a HP, az IBM és az Intel közreműködésével fejlesztett változat egyelőre Intel alapú gépekhez készült el. A Carrier-Grade Linux tervezet olyan Linux-változatok összeállítását célozza, amelyek megfelelnek a távközlési ipar sajátos elvárásainak, és amelyek segítségével szabványokon alapuló, moduláris távközlési rendszerek, szolgáltatások és termékek vezethetők be. Ezt az elgondolást követi az új SuSE-változat, ami felügyeleti és üzleti támogatási rendszerekben, átjárókban, jelkezelő berendezésekben, hang- és adatkezelő rendszerekben és vezeték nélküli megoldásokban egyaránt szerephez juthat.

A SuSE Linux CGL Edition fontosabb jellemzői:

- magas rendelkezésre állást biztosító szolgáltatások, hibakeresési és -javítási, illetve terhelésátvételi lehetőségek;
- IPv6 és mobil IPv6 RFC-k támogatása;
- programból megoldott valós idejű működés, alacsony válaszidők;
- RAID 0-támogatás;
- alkalmazások előtöltése, aminél a rendszer még a futtatás megkezdése előtt lefoglalja a megfelelő memóriaterületeket az alkalmazások számára;

A SuSE Linux CGL Edition ingyenesen, javítócsomag formájában érhető el mindazok számára, akik már rendelkeznek a SuSE Linux Enterprise Server 8-változattal.

A Carrier Grade Linux munkacsoport az Open Source Development Labs egyik csoportja. Az OSDL Linux alapú megoldások fejlesztését ösztönző szervezet, ami gyártóktól függetlenül, non-profit alapon végzi tevékenységét.

➲ http://www.osdl.org
➲ http://www.suse.com

Blu-ray

A Sony bemutatta a világ első Blu-ray Disc előírások szerint működő optikai meghajtóját. A DVD utódjának szánt Blu-ray – kék lézert használó – formátum fejlesztésekor az volt a cél, hogy HDTV minőségű mozgóképet lehessen rögzíteni az újfajta lemezekre – igaz, ez a lehetőség megfelelő műsorszolgáltatás hiányában hazánkban inkább csak vágyálom marad. A fejlesztés befejezettnek egyelőre közel sem mondható, műszaki gondok – és a lehető leghamarabbi megjelenésre való törekvés – miatt a meghajtó az eredetileg eltervezetnél szűkebb szolgáltatáskészletet nyújt, és a költségek csökkentése érdekében számos CD- és DVD-meghajtókból átvett részegységet tartalmaz. A hazánkban valószínűleg jó ideig még amúgy sem beszerezhető készülék megvásárlását érdemes elhalasztani – a Sony gőzerővel dolgozik a különféle formátumok támogatásán, a meghajtót újabb szolgáltatásokkal és egységekkel fogja bővíteni, fejleszteni. Annyit azonban érdemes tudni róla, hogy 23,3 GB-ot tud tárolni lemezenként, folyamatos átviteli sebessége 9 MB/másodperc, és várhatóan Ultra160 SCSI felülettel is meg lehet majd vásárolni.

Adatbázisok apróságoknak

A Solid Information Technology bemutatta a Solid BoostEngine 4.0-t. A BoostEngine egy magas rendelkezésre állású – a Carrier-Grade Linux követelményeket teljesítő – relációs adatbázis-kezelő rendszer, amit elsősorban beágyazott készülékekbe, például útválasztókba, hálózati kapcsolókba, átjárókba szánnak.

A Solid megoldása valójában két, egymással szorosan együttműködő részből épül fel, egy adatbázismotorból és egy adatbázis-összehangoló modulból. Az utóbbi segítségével másolatot lehet fenntartani a kérdéses készüléken futó adatbázisról, majd a másolat felhasználásával azonnali feladatátvételt vagy terheléselosztást lehet végezni, így szinte bármilyen környezetben rendkívül jó rendelkezésre állást lehet biztosítani.

➲ http://www.solidtech.com

Medgyesi Zoltán
(mz@rettesoft.hu)
A Linuxvilág hírszerkesztője. Szabadidejét legszívesebben a barátnőjével tölti, szeret autózni és bográcsban főzni.

Vállalatirányítási rendszerek kicsiben is...

Milyen szerepet vállalnak a minisztériumok?

Manapság a középvállalatok számára egyre fontosabb kérdés az informatika. Nemcsak azért, mert a szervezetet szinte csak ennek segítségével ellenőrizhetik a vezetők, de olyan segítséget is ad, amivel lényegesen gyorsabban és hatékonyabban képesek részt venni a piacon. Az EU-csatlakozás kapcsán több fórumon és több szempontból előtérbe került a vállalati informatika, ezen belül is a vállalatirányítási rendszerek (VIR). Az idehaza felmerülő igényt a minisztériumokban is érzékelik, olyannyira, hogy mind a Gazdasági és Közlekedési Minisztérium, mind az Informatikai és Hírközlési Minisztérium pályázattal kíván javítani a jelenlegi helyzeten.
Ha a célokat nézzük, mindkét pályázat lényege, hogy a pályázó az egész vállalatot átfogni képes rendszert vásároljon, mindkét pályázatra azonos keret (négyszázmillió forint) áll rendelkezésre, valamint mindkét minisztérium a több területet lefedni kívánó pályázatokat részesíti előnyben. Mindkét minisztérium azonban a saját szempontjait szem előtt tartva igyekezett kiírni a pályázatát, ezért a két VIR-pályázat sok mindenben különbözik egymástól. Míg a GKM saját bevallása szerint is elsősorban gazdaságilag megfontolt döntést vár a pályázótól, az IHM nagyon komoly szakmai követelményrendszert állított fel a termék beszállítójával szemben. A pályázók szemszögéből vajon melyik a jobb pályázat?
Nézzünk meg egy-két alapvető különbséget. Talán a legfontosabb, hogy a GKM által kiírt pályázat a pályázó alkalmazottjainak számát kilencnél, míg az IHM 49-nél többen határozza meg. Gondolom, nem egyedül vagyok olyan, aki úgy gondolja, hogy pont a tíz és ötven fő közötti létszámmal dolgozó társaságok számára jelent rendkívüli nehézséget egy VIR beszerzése és üzemeltetése. A tíz fő alatti cégek nagyjából átláthatók, ötven fő fölött pedig fura egy cég az, amelyik nem képes kitermelni egy rendszer költségét. Valahogy olyan érzésem támadt, hogy az IHM által kiírt pályázat kifejezetten megkívánja, hogy a beszállító valamelyik „nagyhal" legyen.
Ugyanezt támasztja alá az IHM által a szállítóval szemben megkövetelt rendkívül sok tulajdonság – hogy ne keressünk sokáig: hány beszállító mondhatja el magáról, hogy a tavalyi év folyamán összértékben több mint százmillió forintban forgalmazott, telepített rendszereket? Vagy hogy fel tud mutatni legalább öt céget, ahol a rendszer már működik (ebből három cégnek legalább 50 fősnek kell lennie)? A további szigorú feltételekkel is nyilván a szakmai igényesség jegyében szigorítja az IHM a kiírást, bár nem tudom, mennyire válik ez a pályázatok előnyére. Valahogy jobban tetszik a GKM hozzáállása: a vállalat döntse el, milyen rendszer jó a számára, de utána legalább öt évig üzemeltesse is!
Az IHM pályázata nagyobb teret enged (legfeljebb 15 MFt), ebből következik, hogy olyan bevezetések pályázását is várják, amelyek összértékben 30 MFt fölé rúgnak. Ezzel szemben a GKM által kiírt pályázat ötmillió forintos felső határa lényegesen kisebb bevezetésekre enged következtetni. Ha tehát a pályázni kívánó cég összköltsége (pontosabban a megpályázható elemek összköltsége) nem haladja meg a tízmillió forintos keretet, mindenképpen a GKM pályázatán érdemes elindulni, figyelembe véve a tágabb lehetőségeket, valamint a másik pályázatban a programmal és a szállítóval támasztott rendkívüli követelményeket. Amennyiben pedig ötven fő fölötti cégnél vagyunk, és valamelyik – minőségbiztosítási rendszerrel is rendelkező – óriásrendszerét kívánjuk megvásárolni, akkor nagy segítség lehet az IHM pályázata.
Mindkét esetben fontos a rendszer hosszú távú költségeit is figyelembe venni. Itt jönnek a varázslatos rövidítések, például a TCO. Ugyanis előfordulhat, hogy támogatással együtt csupán tízmillió forintba kerül számunkra egy négymodulos rendszer bevezetése, de az elkövetkező években „üzembentartás" címszó alatt holmi pármilliós éves költséggel számolhatunk. A másik érdekes kérdés, hogy a fejlesztő (szállító) mennyiért hajlandó továbbfejleszteni a rendszerünket? Ebből a szempontból az IHM pályázata előrelátóan kiköti, hogy a bevezetendő rendszer kapcsán a szállítónak be kell mutatnia, hogy lehetőség van a további bővítésre. Azt viszont nem határozza meg, hogy ezt milyen árszínvonalon kell megtenni. A „röghöz kötés" tehát mindkét esetben komoly veszélynek tűnik.
Egy további érdekes kérdés, hogy miként viszonyul a két pályázat a nyílt rendszerekhez. A GKM elmondása alapján, amennyiben a szállító be tud mutatni egy használható rendszert, annak alacsony költségei még előnyként is szerepelnek az elbírálás során. Az IHM pályázata sajnos a szigorú megkötések kapcsán (magyar támogatás, magyar program, a szállítóval szemben támasztott igények stb.) kifejezetten a nyílt megoldások ellen szól.
Igaz, minél nagyobb a vállalat, annál könnyebben fizet meg egy óriásrendszert. De annál kisebb szüksége van külső segítségre egy ilyen rendszer megvásárlásához! Akárhogy is, örömmel fogadom mindkét minisztérium pályázatát, hiszen azt mutatják, hogy „odafent" is látják, tudják, hogy szükség van segítségre, valamint azt is, hogy a lehetőségek keretein belül meg is kívánják adni ezt a segítséget.

KAPCSOLÓDÓ CÍMEK

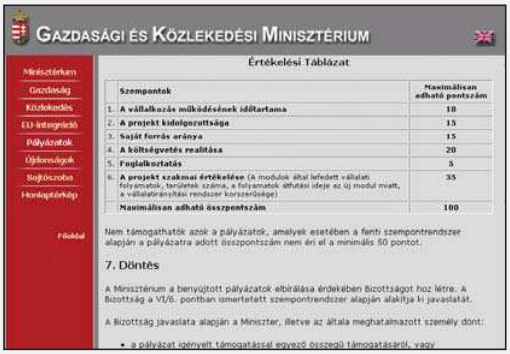
A Gazdasági és Közlekedési Minisztérium pályázatai
➲ http://www.gkm.hu/gk/index_.paly
Az Informatikai és Hírközlési Minisztérium VIR-pályázata
➲ http://www.ihm.hu/tarsadalom/palyazatok/ihm/ihm_20030508_1.html

Szy György (Szy.Gyorgy@linuxvilag.hu)
Nap mint nap saját bőrén érzi, hogy mennyire fontos az informatika egy vállalatnál

Vállalatirányítási rendszer GKM módra

A Gazdasági és Közlekedési Minisztériumon belül a vállalatoknál használandó vállalatirányítási rendszerek (VIR) kapcsán elsősorban nem a számítástechnikai kérdések a fontosak. A Minisztérium célját – a pályázó vállalatok hatékonyságának növelését – a pályázóval együtt kívánja elérni, sőt megbízik a vállalat programválasztásában, hiszen minden vállalat más és más igényeket támaszt egy vállalatirányítási rendszerrel szemben.
A Gazdasági és Közlekedési Minisztérium összértékben 400 millió forintot szánt a vállalatirányítási rendszerek beszerzésének, üzembe helyezésének támogatására.
Magyarné dr. Szabó Krisztinát, a GKM Kis- és Középvállalkozás-stratégiai és Pályázati Főosztályának főosztályvezetőjét kérdeztem a Széchenyi Vállalkozásfejlesztési Program „A kis- és középvállalkozások részére a korszerű vállalatirányítási rendszerek támogatására (SZVP-2003-2)" címen kiírt pályázatával kapcsolatban.

Szy György: Milyen megfontolások alapján döntött úgy a minisztérium, hogy külön pályázatot ír ki a vállalatirányítási rendszerek támogatására, és mit nyerhetnek a pályázók?
Magyarné dr. Szabó Krisztina: A minisztérium számára nagyon fontos, hogy a megcélzott kis–közép blokkban minél hatékonyabb és egységesebb informatikai rendszerek működjenek. Ez rendkívül fontos a mai kiélezett piaci helyzetben, és különösen nagy szerepet kap az elkövetkező években. A pályázat kiírásánál külön fontosnak tartottuk, hogy ne az informatikai piac néhány szereplőjét támogassuk, hanem anyagi segítséget adjunk az egységes rendszert használni kívánó vállalatok számára. Pályázatokat tehát a rendszert vásárolni kívánó vállalatoktól várunk. Emellett fontosnak tartjuk, hogy a vállalat saját maga döntse el, hogy milyen rendszer felel meg a céljainak, hiszen ahány tevékenységi kör, annyi különböző igény. Megfontolt döntésre ösztönöz a negyven százalékos támogatási korlát, valamint az is, hogy a bevezetett rendszert a támogatás lezárása után öt évig üzemben (és szinten) kell tartani.
Sz. Gy.: Mely részét támogatják pontosan egy ilyen tervezeten belül? Gondolok itt arra, hogy a helyzetfelmérésre, a bevezetésre vagy az üzembentartáshoz is pályázható-e támogatás? Gyakran egy-egy vállalat igénye oly sajátos, hogy a vállalatok maguk készítik a rendszerüket. Támogatják-e a belső fejlesztéseket?
M. Sz. K.: Teljes bevezetéseket, sőt teljes csomagokat is lehet pályáztatni, a helyzetfelméréstől a bevezetésig (az üzembentartásra a pályázat nem vonatkozik), így a csomagba akár egyéb kapcsolódó költségek is beletartozhatnak. Fontos kiemelni azonban, hogy a pályázatnak nem elsődleges célja, hogy annak révén a vállalatok belső bérköltségeit vagy például a gépparkfejlesztését támogassuk. Elsősorban külső cég által szállított rendszer használatát támogatjuk, ezt a számlaalapú elszámolási rendszer is mutatja.
Sz. Gy.: Hogyan viszonyulnak a nyílt rendszerekhez? Van-e megkötés a pályázott termékek szállítója kapcsán?
M. Sz. K.: Mint mondtam, fontosnak tartjuk, hogy a vállalat komolyan átgondolja, milyen rendszert kíván használni – ez a saját érdeke is, hiszen a költség nagyobbik részét ő állja. Ennek fényében, ha egy vállalat nyílt rendszer használata mellett dönt, és a választott rendszer ugyanúgy kielégíti a vele szemben támasztott követelményeket, ez nem befolyásolja a pályázat elbírálását. Sőt például a frissítési, illetve követési díjak mértéke esetleg egy nyílt rendszer felé is billentheti a mérleg nyelvét.



Sz. Gy.: Hogyan történik egy-egy pályázat elbírálása?
M. Sz. K.: A pályázatokat egy bizottság bírálja el, amelyet független szakemberekből állítottunk össze. Az elbírálás során a pályázatnak egy adott pontozás alapján el kell érnie legalább az ötven százalékot. Természetesen a pályázat elbírálásához tartozó ponttáblázat nyilvános, ez alapján látható, hogy a pályázat akkor is kaphat támogatást, ha a bizottság nem ad teljes pontértéket például a projekt kidolgozottságára.
Sz. Gy.: Tehát ha jól értem, jó eséllyel indulhat a pályázaton egy külső cég által bevezetendő termékre alapozott projekt, akkor is, ha a termék nyílt forráskódú? Például egy teljes csomag keretében, ahol a csomagba a helyzetfelméréstől kezdve a bevezetésen és betanításon át egészen az éles rendszer elindításáig minden beletartozik?
M. Sz. K.: Igen, természetesen. A mi célunk, hogy a vállalatot segítsük saját üzletmenetének javításában, így nem a döntést kívánjuk meghozni helyette. Bízunk benne, hogy a pályázó képes megítélni, hogy az általa választott termék biztosítani tudja-e számára hosszú távon a hatékony működést.
Sz. Gy.: Hova fordulhatnak további kérdéseikkel az érdeklődők?
M. Sz. K.: Lehetőség nyílik az érdeklődők számára, hogy a benyújtandó pályázat kapcsán munkatársainkkal a 06-40-630-530-as kék számon konzultáljanak. Emellett kérdéseikkel a Minisztérium weboldaláról letölthető anyagokon túl a Magyar Vállalkozásfejlesztési KHT-t is felkereshetik.
Sz. Gy.: Köszönöm a tájékoztatást.

Szy György (Szy.Gyorgy@linuxvilag.hu)

A minisztériumi pályázatok rövid összevetése

Intézmény	GKM	IHM
Pályázat célja	Egységes VIR, modulok összekötése.	Korszerű VIR, a létező korszerűsítése.
Pályázható elemek	Vásárlás, bevezetés – gép, program, rendszerdíj. Elbírálás alapján teljes csomag is!	Vásárlás, bevezetés – legalább három modul bevezetése. Ezen belül: beszerzés, fejlesztési díj, licensz, know-how, tanácsadás, rendszerterv, oktatás, telepítés, üzembe állítás, tesztelés.
Nem támogatandó	Tanácsadás, előkészítés, dokumentációs költségek, reprezentációs költségek, saját alkalmazott bére, közterhek.	A költséghatékonyság elvét megszegő tevékenységek, a piacon általánosan elfogadottnál nagyobb összegek.
Pályázók köre	Gazdasági társaságok, szövetkezetek, egyéni vállalkozók.	Gazdasági társaságok.
Létszámmegkötés	9 és 250 fő között.	49 és 250 fő között.
Pénzügyi megkötés	Nettó árbevétel < 4 Mrd Ft Mérlegfőösszeg < 2,7 Mrd Ft	Nettó árbevétel < 4 Mrd Ft, de > 700 M Ft. Mérlegfőösszeg < 2,7 Mrd Ft, de > 500 MFt
Szükséges önerő, biztosíték	Legalább 25% önerő kell (nem hitel és nem tagi kölcsön!). Biztosíték nem feltétel.	Legalább 25% önerő szükséges (ami nem hitel és nem tagi kölcsön!).
Kizáró ok	Mezőgazdasági tevékenység, veszteséges gazdálkodó vagy negatív tőke.	
Pályázati díj	Az igényelt támogatás 0,2%-a, de legalább 1000 Ft.	Egyszeri 30 000 Ft.
Benyújtandó különleges dokumentumok	Nincs.	Teljes tevékenységre kiterjedő vállalati üzleti terv legalább 3 évre előre, ágazati elemzés, részletes elemzés a jelenlegi rendszerről.
A rendszer szállítójával szembeni követelmények	A vállalkozó tevékenységét, illetve területét segítse (bizottság dönt).	A rendszer szállítójának be kell nyújtania legalább öt referencialevelet hazai vállalatoktól (ebből háromnak a pályázatban feltüntetett feltételeknek meg kell felelnie). Rendelkeznie kell felkészült szakembergárdával. Az általa az elmúlt három év során leszállított rendszerek összértékének el kell érnie a > 100 MFt értéket. Színvonalas szakmai szolgáltatás, bővítési, követési lehetőségek bizonyítása. Minőségbiztosítási rendszerrel szükséges rendelkeznie.
Egyéb szakmai követelmények	Nincs.	4GL, relációs adatbázis, modularitás, azonnali adatfeldolgozás, bővíthetőség, kiszolgálóalapú vagy webes felépítés. Operációs rendszer: Linux, MS vagy UNIX, magyar változat, magyar dokumentáció, magyar támogatás, működő referenciák. Minden modullal szemben részletes elvárási lista.
Beadási határidő	Szeptember 30-ig vagy közleményig.	Közleményig (első elbírálási kör: június 30-ig, beadottak július 31-ig).
Nyertesek várható száma	100–110	
Rendelkezésre álló keret	400 MFt	400 MFt.
Támogatás lehetséges mértéke	40%, de legfeljebb 5 MFt (h. h. megyékben 50%, 6,25 MFt).	50%, de legfeljebb 15 MFt.
Elbírálás	Bírálóbizottság, nyilvános pontrendszer alapján.	Bírálóbizottság. Elemzett területek: a kínált szolgáltatások, a benyújtott referenciák, a pályázó piaci helyzete, a rendszer várható hatása.
Megvalósítási időszak	Két éven belül	A lehető legrövidebb időn belül, de legfeljebb 10 hónapon belül.
Üzembentartás	Megvalósítás után legalább öt évig.	Három évig.

A programszabadalmakról

Híradások számoltak be róla, hogy a szabad programokat preferáló szervezetek egy nyolcfős küldöttséget menesztettek Brüsszelbe, ami egy meghallgatáson vett részt az Európai Parlamentben. Aki jól figyelt, még a meghallgatás szervezőjének köszönetét is láthatta, aki sikeresnek értékelte azt. A kérdés már csak az, hogy a programszabadalmak ellen miért kell tiltakozni?

A szabadalmaknak bizonyos területeken van létjogosultságuk. A gyógyszerek esetén például lehetővé teszik a cégek számára, hogy a kutatásra és fejlesztésre költött hatalmas összegeket ismét visszaszerezhessék. Szintén alkalmas arra, hogy az új módszereket közzétehessük, anélkül, hogy valaki ellopná őket. A szabadalmak a bonyolult kutatások eredményeinek egyszerű használatától védik meg a fejlesztőt. A versenytárs számára egyszerű ismert orvosságokat előállítani, a nehéz a dologban a recept kifejlesztése.

A programoknál ennek az ellenkezője igaz: a „kutatások" egyszerűek, de a végeredmény lemásolása nehéz. Egyszerű ötleteket kitalálni a továbbfejlesztésre, vagy arra, hogy a számítógépet hogyan használjuk újfajta módokon, ezt bárki meg tudja tenni. A neheze a programok tényleges megírása. Csaknem olyan nehéz lemásolni egy program működését (nem a programot magát), mint az eredetit megírni. Ezért elegendő a programok védelmére a szerzői jog. A szerzői jog azt védi, aminek az előállítása a legtöbb időt, energiát és elkötelezettséget igényli.

Minden program egyszerű szövegből áll, valamilyen programozási nyelven megírva. Ezt a szöveget – amit forráskódnak nevezünk – egy fordítóprogram segítségével alakítjuk át gépi kódra. Ezt egy egyszerű ember minden különösebb erőforrás nélkül meg tudja tenni.

A program egy olyan termék, amelyik tudásból, képzeletből és elkötelezettségből készül, hasonlóképpen, mint egy festmény vagy bármely műalkotás. Amikor szabadalmaztatunk, a tudás közkinccsé válik az egyes felhasználók, az iskolák és a társadalom hasznára. A programok esetében ez nem lehetséges ugyanolyan módon, hiszen a tervrajz a forráskód. A program önmagát tükrözi, a leírása saját maga. A forráskód közzététele ugyanaz, mint magának a programnak a továbbadása, az olvasónak csak le kell másolnia a szöveget és a saját számítógépén használnia.

Az algoritmusok, matematikai formulák és programötletek szabadalmazása az innováció szempontjából butaság, nem segíti elő a fejlődést. Éppen ellenkezőleg, lehetetlenné teszi a programfejlesztést anélkül, hogy a fejlesztő meg ne sértsen valami már szabadalmazott ötletet, ráadásul a tudtán kívül téve ezt. A fejlesztők számára csaknem lehetetlen újat létrehozni, anélkül, hogy valami már szabadalmazott dolgot újra fel ne találnának. Ha valaki ezzel komolyan próbálkozna, annyira különös és szokatlan programokat kellene írnia, hogy senki sem akarná használni őket.

A programszabadalmakkal csak a szabadalmi jogászok és a nagy multinacionális programkészítő óriások járnának jól, mert akkora készletük van a szabadalmakból, hogy mindig találhatnak egy szabadalmat, amivel beperelhetnék a versenytársaikat, és díjakat szedhetnének tőlük, amint feltűnnek a porondon.

Említettem, hogy a programok fejlesztése gyakorlatilag lehetetlen anélkül, hogy valamilyen már szabadalmaztatott ötletet megsértenénk. Felmerülhet a kérdés, hogy ennek mi az oka. A programok sokkal összetettebb alkotások, mint más, a szabadalmak hatálya alá tartozó területek. Lássunk egy példát ennek megértésére!

Mi történt volna akkor, ha Európa királyai az 1700-as években úgy döntenek, hogy a zene fejlődését elősegítendő, a zenei ötletekre szabadalmakat lehet bejegyezni? Valószínűleg ma szegényebbek lennénk nagyon sok zeneművel, például Beethoven szimfóniáival is. Egy zeneszerző a komponálás során nemcsak a saját zenei ötleteit használja fel, hanem a már meglévőkből is válogat. A zene nem légüres térből születik, hanem az emberi kultúrára épít. Beethovennek, ahhoz hogy a saját zenei ötleteiből nagyszerű szimfóniákat alkosson, építenie kellett az előtte járó zeneszerzők által felhalmozott zeneelméleti tudásra. Pontosan ezt akadályozták volna meg számára a szabadalmak. Mindig is léteztek kísérleti zeneszerzők, akik kizárólag a saját ötleteiket használták fel; az általuk írt zenét nagyon kevesen képesek meghallgatni. Egy szimfónia bonyolult szerkezet, és csak akkor szól teljes pompájában, ha minden a helyén van. A programoknál ugyanez a helyzet. Ma bármely bonyolultabb program több bejegyzett programszabadalomban leírt ötletet is felhasznál. Európában a programszabadalmak törvénytelenek. Még az a negyvenezer is, amelyeket az Európai Szabadalmi Hivatal már bejegyzett. Legalábbis egyelőre.

Jó, jó, mondhatjuk, hogy hiszen az Egyesült Államokban már régóta vannak programszabadalmak és mégsem omlott össze a gazdaságuk, mi félnivalónk van nekünk a szabadalmaktól?

Csupán elgondolkodásra ajánljuk: az amerikai cégek a szabadalmaikat csak ritkán alkalmazzák. Ez igaz is. De egy hibás szabály meghozatalára nem indok az, hogy hátha nem fognak visszaélni vele.

Magosány Árpád (mag@bunuel.tii.matav.hu)

KAPCSOLÓDÓ CÍMEK

Az FSF.hu kapcsolódó oldala
➲ http://www.fsf.hu/index.php/szabadalmak
Néhány bejegyzett szabadalom
➲ http://www.linux.hu/article.php?id=1695
Szervezeteink legutóbbi közleménye
➲ http://www.linux.hu/article.php?id=1733
Az EuroLinux néhány kapcsolódó elemzése
➲ http://swpat.ffii.org/papers/eubsa-swpat0202/pref/index.en.html
➲ http://swpat.ffii.org/papers/eubsa-swpat0202/esse/index.en.html
Ellenjavaslat
➲ http://swpat.ffii.org/papers/eubsa-swpat0202/kern/index.en.html

Az Ericsson közzétette a TIPC forráskódját

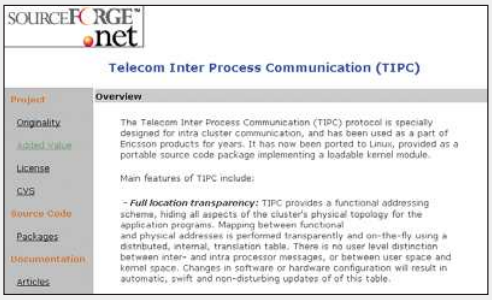
Az Ericsson 2003. február 3-án a GNU GPL szerint közzétette a TIPC (Telecom Inter-Process Communication – Adatátviteli Folyamatközi Kapcsolattartás) forráskódját a nyílt forrás közössége számára. A TIPC egy, a telepeken belüli kapcsolattartásra külön megtervezett protokoll, amit az Ericsson termékek részeként évek óta szerte a világon telephelyek százainál használnak. Most elkészült a linuxos változat is, mégpedig betölthető magmodul formájában.

A TIPC hasznos eszköztár bárki számára, aki Carrie-Grade Linux-telepeket kíván fejleszteni vagy használni. A telep, a hálózat és a rendszer irányításához szükséges hátteret adja. A felhasznált címzési módszer egyedülállónak tűnik az indexelési szolgáltatások és a gyors kapcsolatteremtés terén. Előnye továbbá, hogy a jelzésváltás megvalósítása bármilyen hordozón teljes terhelésmegosztást és megbízható hibatűrést biztosít.

A TIPC jellemzői:

- Öntanuló helymeghatározás: a TIPC olyan címzési módszerrel bír, ami a telep fizikai felépítését minden tekintetben elrejti az alkalmazások elől. A felhasznált és a fizikai címek összerendelése öntanuló módon zajlik, működés közben egy osztott belső fordítótáblát használ.
- Egyszerűsített, „fürge” kapcsolatteremtés: az egy tranzakción belüli üzenetváltás – beleértve a kapcsolatlétesítést, a rövid adatátvitelt és a leállítást – testreszabható, így még hatékonyabbá tehető, ráadásul rejtett protokollüzenetek nélkül. Egy már létrejött kapcsolat visszajelez, és bármilyen szolgáltatási hibáról jelentést készít az alkalmazásnak.
- Általános, alkalmazkodóképes, jelzéskapcsolati protokoll: olyan, jellegzetesen a szállítási rétegen megvalósított feladatok, mint az adásismétlés, a szakaszolás (segmentation), a csomagkészítés és a folytonosság-ellenőrzés lekerült a jelzéskapcsolati rétegre. Emiatt ez a réteg ugyan összetettebbé válik, viszont jobb erőforrás-kihasználást tesz lehetővé, és sokkal hatékonyabb veremkezelést eredményez. A jelzéskapcsolatok állítható frekvenciájú folytonossági ellenőrzés szigorú felügyelete alatt állnak, emellett képesek megállapítani és közölni a jelzéshibákat egy másik kapcsolat részeként. A fölösleges kapcsolatok hibatűrése ilyen esetekben öntanuló módon és zavarmentesen zajlik. A jelzéskapcsolatok önbeállítók, lehetőség szerint üzenetszórásos, illetve csoportos üzenetszórásos szomszédfelismeréses protokollt használnak.
- Teljesítmény: a TIPC a rövid (< 1 KB) üzeneteket a TCP/IP-nél 25–35 százalékkal nagyobb, a hosszabbakat pedig összemérhető sebességgel szállítja a processzorok között, a processzoron belüli kézbesítés pedig 75 százalékkal gyorsabb. Ezen túlmenően az egyszerűsített kapcsolódást használva egy tranzakció legkevesebb két üzenetváltással megvalósítható, szemben a TCP/IP-vel, ahol legalább kilencre szükség van. Emiatt a távközlésben használatos rövid tranzakciók a megfelelő TCP-tranzakciók tört része alatt elvégezhetők.
- Szolgáltatásminőség: a sorrendi, veszteségmentes üzenettovábbítás kapcsolatközpontú és kapcsolatnélküli módban egyaránt szavatolt. A célállomás elérhetetlensége esetében a nem kézbesített üzenetek egy hibakóddal együtt visszakerülnek a küldőhöz, ez a kód utal a hiba okára.
- Indexelési lehetőség: az alkalmazások sorszámozhatják a felhasznált és a fizikai címek elérhetőségét, illetve elérhetetlenségét. Ez azt jelenti, hogy a telepen történt működési vagy felépítési változások nyomon követése egyszerű, akárcsak az osztott alkalmazások indításának összehangolása.

➲ http://www.osdl.org/projects/cgl



➲ http://tipc.sourceforge.net

Azt tervezzük, hogy az elkövetkező hónapokban teljes műszaki leírást készítünk a TIPC-ről a Linux Journalba; addig is írjanak bátran Jon Maloynak (Jon.Maloy@Ericsson.com), hogy bármilyen, a TIPC-t érintő kérdést megvitathassunk.

Jon Maloy és *Ibrahim Haddad*

Linux Journal 2003. május, 109. szám

KAPCSOLÓDÓ CÍMEK

Carrier Grade Working Group
➲ http://www.osdl.org/projects/cgl
Open Source Development Lab ➲ http://www.osdl.org
TIPC és SourceForge oldal ➲ http://tipc.sourceforge.net

Oktatás, támogatás, dokumentáció

Beszélgetés az ULX Kft., a SuSE Linux AG magyarországi irodájának munkatársával, Hilzinger Marcellal.

Csontos Gyula: Az első kérdésem az lenne, hogy mióta dolgozol a Linux népszerűsítésén e cég keretein belül, és mik a feladataid? Mesélj magadról néhány mondatot!

Hilzinger Marcel: A zürichi tudományegyetemen végeztem 1999-ben. Itt szereztem német nyelv és irodalomból, politológiából és kelet-európai történelemből oklevelet. Az iroda megalakulása óta dolgozom itt, a feladataim röviden: a dokumentáció karbantartása, ami a fordítást, a frissítést, az összehangolást is magában foglalja. Másik két nagy feladatköröm az oktatás és a terméktámogatás megszervezése.

Cs. Gy.: *Mesélj a feladataidról egy kicsit bővebben, kezdjük először az általad említett dokumentációval.*

H. M.: A SuSE dobozos változatához adott kézikönyvek fordításával kezdődött minden. Az első fordításoknál körülbelül ötven–ötven százalékos arányban fordítottuk külső segítséggel a könyveket, most már 95 százalékban saját fordításokról van szó. A könyvek fordításának nagyobbik része az eredeti német nyelvű dokumentációból közvetlen fordítással kerül ki, a kisebbik része pedig angol fordításokból érkezik, a munkatársak közül mindenki részt vesz a dokumentáció létrehozásában. Az anyag LaTeX-ben készül, de a magyar nyelvi sajátosságok miatt előbb egy foltot kellett készíteni a Linuxban szereplő LaTeX rendszerhez, ami már belekerült a LaTeX hivatalos forrásfájába is. A dokumentációkat jelenleg Magyarországon készítjük és gyártjuk, ellentétben a régebbi gyakorlattal, amikor mindent Németországban nyomtattak.

Cs. Gy.: *A dokumentációkat az üzleti termékekhez is elkészítitek? Egyáltalán, mennyire várják el az üzleti felhasználók a magyar nyelvű kézikönyveket?*

H. M.: Egyenlőre még nincs minden üzleti termékhez teljes magyar nyelvű dokumentáció, azonban dolgozunk rajta. Minden Magyarországon forgalmazott termékhez szükséges magyar nyelvű „használati utasítást" adni, az indíttatás mégsem ez volt arra, hogy üzleti termékeinket is ellássuk magyar nyelvű kézikönyvvel, hanem az, hogy a felhasználóink igenis igénylik és el is várják az efféle szolgáltatást.

Cs. Gy.: *Milyen egyéb feladatokat ró rád, rátok a dokumentáció karbantartása?*

H. M.: Azokról a programokról, amelyekről úgy gondoljuk, hogy érdemes részletesebb leírást készíteni, a dokumentációt mi magunk készítjük el, és a dobozban található könyvek szerves részeként, fejezeteként vagy a már meglévő fejezetrészként jelentetjük meg.

Cs. Gy.: *Említetted a beszélgetés elején az oktatást; milyen képzéseket, tanfolyamokat tartotok?*

H. M.: Tanfolyamaink honosított, hivatalos SuSE Linux-tanfolyamok. Jelenleg a tanfolyamok szerkezetében változások várhatók (a mellékelt ábrán áttekinthető e felépítés), mint látható, nagymértékben épít a világon mindenhol ismert és elfogadott LPI-tanfolyamok anyagaira, természetesen kiegészítve ezt a UnitedLinuxhoz tartozó anyagokkal.

Cs. Gy.: *Az oktatást csak ti végzitek vagy partnereitek is vannak?*

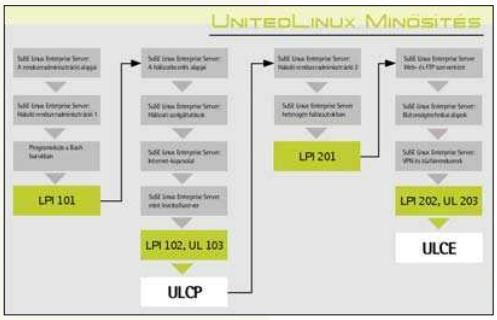
H. M.: Jelenleg országszerte tíz hivatalos oktatási partnerünk van. Oktatással foglalkozó partnereinknek igen magas követelményrendszernek kell megfelelniük. Megfelelő gépteremmel, projektorral és egyéb felszereléssel kell rendelkezniük ahhoz, hogy a minőségi oktatást meg tudják valósítani.

Cs. Gy.: *Csak a ti és partnereitek oktatási bázisán végeztek oktatást, vagy kihelyezett tanfolyamokra is van példa?*

H. M.: Természetesen, ha egy ügyfél azt szeretné, kihelyezett oktatást is vállalunk, amit akár az ügyfél telephelyén is meg tudunk tartani.

Cs. Gy.: *A vizsgáztatás hol és milyen nyelven folyik?*

H. M.: Minden elvégzett tanfolyamról kiállítunk egy hivatalos magyar nyelvű tanúsítványt. Az LPI- és a UnitedLinux-vizsgákat a hivatalosan elismert VUE központokban lehet letenni, angol nyelven.



Cs. Gy.: *Milyen a támogatási rendszeretek?*

H. M.: Minden termékünkhöz jár a 30 napos telepítési segítség faxon, telefonon és elektronikus levélben. Ez, mint a neve is mutatja, csak a rendszer telepítéséhez nyújt segítséget. Ezen kívül jár egy év aktív programkarbantartás is. Mivel nagyon sok kérés érkezik be hozzánk, ennek felügyeletét egy központi rendszeren végezzük, itt nyomon követhető, hogy ki, mikor és mit válaszolt egy támogatási kérdésre. Az emberek is nagyon sokfélék, a gondjaik pedig még sokszínűbbek, mint azt elsőre el tudnánk képzelni, ezért szükség van egy házirend felállítására és betartására. Van, aki kevesli ezt a támogatást, azonban az általunk nyújtott segítség sok ponton túlmegy a hivatalos telepítési segítségen.

Cs. Gy.: *Mi a helyzet a üzleti termékeitekkel, azokhoz milyen támogatást adtok?*

H. M.: Ezekhez a termékekhez alapból szintén a 30 napos telepítési támogatás jár, illetve szintén vásárolható hozzájuk külön támogatás. Továbbá egyéves támogatást, öt év programkarbantartást, ami a különféle programhibák javításait is magában foglalja, valamint pluszban jelszavas hozzáféréssel ellátott terméktámogatási adatbázist is nyújtunk hozzájuk.

Cs. Gy.: *Hogy a munkán kívül másról is szó essen, mivel töltöd a szabadidődet a legszívesebben?*

H. M.: Hasznos multimédiás programokat magyarítok, bakelit meselemezeket digitalizálok a most három hónapos kisfiamnak későbbi felhasználásra, és részt vettem az OpenOffice.org magyarításában is.

Cs. Gy.: *Köszönöm a beszélgetést!*

Csontos Gyula

Opera magyarul

Rövid internetes pályafutásom alatt sikerült néhány böngészőt kipróbálnom. Mint általában a kezdők, én is az Internet Explorer időszerű változatával kezdtem, azután jött a Netscape 4.x-es sorozata. Amikor Linuxra váltottam, meglévő tapasztalataim folytán szintén a Netscape-et használtam, majd következett a Mozilla és a Konqueror. Mindegyiknek voltak jó és rossz tulajdonságai, mindet másért szerettem. Mégis úgy éreztem, hogy valami mindegyikből hiányzik. Amikor feltettem az Opera legfrissebb változatát, néhány órás állítgatás és ismerkedés után azt mondtam: ez az, ezt kerestem! Saját ízlésünk szerint átszabhatjuk a felületét, saját bőröket és gombokat készíthetünk hozzá, egyszerre több lapon is böngészhetünk (mint a Mozillában), és szinte minden szolgáltatáshoz létezik gyorsbillentyű. A leginkább megragadó számomra mégis az egérhez rendelhető számtalan szolgáltatás volt:

- előre- és hátralépkedhetünk a lapok között,
- ablakokat csukhatunk le, nyithatunk fel és csukhatunk be,
- szövegeket nagyíthatunk és kicsinyíthetünk stb.

Egyetlen gond volt csak a programmal: nem beszélt magyarul. Az Opera honlapján 12 támogatott nyelvet találtam, sajnos a miénk nem volt a listán. Viszont letölthető volt az eredeti üzeneteket tartalmazó fájl, amit le lehet fordítani. Innen már „csak" egy kis gépelés és szótárforgatás volt hátra, és elkészült a próbaváltozat. Ezt nagyjából 10–15 érdeklődő kezdte el használni, akik folyamatosan küldték a hibajelentéseket. Eközben vetette fel valaki, hogy nem ártana a Súgót és a Nap tippjét is lefordítani. Mivel úgyis „benne voltam" a dologban, belevágtam. Ennek eredményeként készült el a teljes szövegnek édes anyanyelvünkre történő átültetése. Mire ez az írás megjelenik, reményeim szerint már az összes hibát sikerült kijavítani, és tökéletes változatot adunk közre.
Természetesen egy ilyen munkát csaknem lehetetlen egyedül elvégezni, nekem is sokan segítettek a hibakeresésben és a javításban.

Köszönetnyilvánítás

Ezúton is szeretnék köszönetet mondani mindenkinek a segítségért, különösen *Sallai András*-nak, aki a gyorsbillentyűk ismétlődéseit vadászta, *Fazakas Albert*-nek, aki a legtöbb elgépelést és vesszőhibát felfedezte, a *„kisbetű"* becenevű ismeretlennek, aki a nyelvtani és stilisztikai hibákat gyomlálta, *Lakos Imré*-nek, *Kő László*-nak, *Rátonyi Gábor Tamás*-nak és *Szabó Zoltán*-nak, akik a terjesztésben segítettek.

A fordítás zipfájlban a következő címekről tölthető le:
➲ http://linuxace.freeweb.hu
➲ http://www.mylinux.hu
➲ http://www.hwsw.hu
➲ http://www.osb.hu/tech/forditasok/Opera
➲ http://www.linuxportal.hu/letoltes/Opera.zip

Mlatilik Zsolt (dr-mac@dunaweb.hu)

Linux-tábor

2003. június 29. és július 12. között idén harmadjára kerül megrendezésre a Linux-tábor. A 2001 nyarán életre hívott, hagyományosan Szerencsen tartott tábor szervezői két fő célt tűztek maguk elé: a Linux-tudás gyarapítását és kellemes pihenést nyújtani a résztvevőknek. A tábor székhelye idén, mint az előző két évben is, a Szerencsi Középiskolai Kollégium, ahol a szállást és az ellátást kapják a résztvevők, míg a tanfolyam a kollégium szomszédságában lévő gimnáziumban zajlik.
Az oktatás három, igény szerint különböző (általában kezdő, haladó és profi) szinten folyik. Az oktatók (akik ingyen vállalják a tanfolyamok megtartását a táborban) Magyarország legjobb linuxos szakemberei közül kerülnek ki. A 2001-es első táborban három 15 fős csoportban folyt az oktatás, míg idén a tavalyihoz hasonlóan három 20 fős csoportban. A szervezők 2002-ben próbálkoztak meg először két önálló turnus szervezésével, és a nagy sikert látva idén is két hét áll a jelentkezők rendelkezésére. Az oktatás a gimnázium számítógéptermeiben folyik, így minden hallgató önálló számítógép mellett gyakorolhat. Emellett lehetőség nyílik saját gép használatára is, amire a szakemberek segítenek feltelepíteni a Linuxot. A délelőtti oktatás után délután számos lehetőséget kínálnak a szórakozásra. A számítógépüktől elszakadni nem tudók beülhetnek a kollégium internetkapcsolattal is rendelkező géptermébe. Újdonság idén, hogy a Magyar BSD Egyesület jóvoltából délutánonként a BSD operációs rendszerrel ismerkedhetnek az érdeklődők. Akik ekkor nem informatikai programra vágynak, szervezetten látogathatják majd meg a szerencsi vár múzeumát, illetve lehetőség lesz csoportos kirándulásokra, és strandot is találunk Szeencs közelében. Az esti program pincelátogatás, ahol egy helyi szőlőgazda mutatja be borait, azaz a Tokaj-hegyvidék nedűit. Az i-re a pontot az egyik este pincevacsora, majd az utolsó esti bográcsozás teszi fel.
A tábort szervezői önköltségi alapon rendezik, így igen olcsón lehet részt venni rajta. A hatnapos turnus napi háromszori étkezéssel, tanfolyami részvétellel, borkóstolással, pincevacsorával felnőtteknek mindössze 24 000 forint, amiből az LME tagjai 2000 forintos kedvezményt kapnak. Ugyanez 16 éves korig csak 18 000 forintba kerül. A táborról tájékoztatás és jelentkezési lap a ➲ http://linuxtabor.hu weboldalon érhető el, a szervezőkkel pedig a linuxtabor@webhome.hu címen léphetünk kapcsolatba.

Linux-index

1. Legkevesebb ennyi ezer dollár az ára az SGI új csúcstermékének, a Linuxot futtató SGI Altix 3000 kiszolgálónak: 30
2. Ennyi millió dollár az ára a legdrágább SGI Altix 3000-es Linux-kiszolgálónak: 1
3. Ennyi régi SGI-gépet cseréltek le Linuxot futtató Dell-gépekre a Sony Pictures Imageworks vállalatnál: 600
4. A Sams Club webáruházban eladásra kínált asztali Linux-rendszerek száma: 1
5. A Sams Club webáruházban kapható asztali Linux-rendszer ára: 297,95 dollár
6. A Wal-Mart webáruházban eladásra kínált különböző (Microtel) Linux-rendszerek száma: 33
7. Lindows alapú rendszerek száma: 15
8. Mandrake alapú rendszerek száma: 9
9. Lycoris alapú rendszerek száma: 9
10. Ennyibe kerül a legolcsóbb (Lindows) Linux-rendszer a Wal-Mart webáruházban: 199,98 dollár
11. Ennyibe kerül a legdrágább (Mandrake) Linux-rendszer a Wal-Mart webáruházban: 648 dollár
12. A japán kormány tervei szerint ennyi ezer dollárt költ a következő pénzügyi évben annak tanulmányozására, hogyan térhet át Linuxra a saját számítógépein: 416
13. Ennyiedik helyen áll a Running Linux az O'Reilly kiadó könyvsikereinek listáján: 1
14. Ennyi példány kelt el a Running Linux című könyvből: 200 000
15. Jelenleg legkevesebb ekkora a Linux-kiszolgálók eladási aránya a Meta Group szerint: 15%
16. Jelenleg legfeljebb ekkora a Linux-kiszolgálók eladási aránya a Meta Group szerint: 20%
17. A Linux részesedése a kiszolgálóeladásban 2006-ra vagy 2007-re a Meta Group szerint: 45%

Források
1–3.: Los Angeles Times
4–5.: ➲ http://samsclub.com
6–11.: ➲ http://walmart.com
12.: Associated Press
13-14.: *Russel J. T. Dyer* Running Linux in a New World
➲ http://www.linuxjournal.com/article/6617
15–17.: Meta Group, Inc.

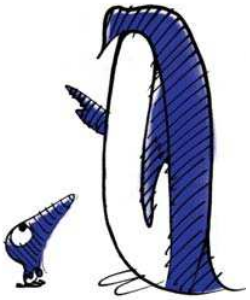
Linux Journal 2003. május, 109. szám

diff -u: rendszermagfejlesztési hírek

Richard Gooch műve, a `devfs` fájlrendszer valószínűleg kimarad a rendszermagból. 2002. december végén *Adam J. Richter* bejelentett egy javítófájlt, ami egy, a RamFS fájlrendszeren alapuló új mechanizmussal váltaná fel a devfs-t. Az új rendszer több téren is utánozni próbálta a devfs viselkedését, habár Adam nem akarta a devfs összes szolgáltatását megvalósítani a RamFS rendszerben. Programját részben a devfs-felület letisztult változatának szánta, hogy azokat a szolgáltatásokat, amiket alig pár rendszer használt, más módszerekkel lehessen helyettesíteni. A szerkezeti változtatások eredményeképpen az eredeti kód méretét a negyedére sikerült csökkentenie. A devfs fájlrendszer mindig is vitatott volt, *Linus Torvalds* döntése pedig, amellyel bevette a devfs-t a hivatalos fába, még több vitát kavart.
A `sysfs` fájlrendszer várhatóan felváltja a */proc* fájlrendszert és a többi olyan módszert, amivel a rendszerinformációk lekérdezhetők. A sysfs eredetileg a vezérlőprogramok íróinak eszköze volt, 2002-ben azonban a rendszermag más részeire is kiterjedt a használata. Azóta számos más felületről folyamatosan próbálnak áttérni a sysfs-re. 2003 januárjában éppen a */proc/cpufreq* fejlesztésén dolgoztak, amikor *Dominik Brodowski* érvénytelennek nyilvánította azt, s helyette a sysfs felületet vette be a `cpufreq` kódjába. *Patrick Mochel* ugyancsak közreműködött a kód átfésülésében, ellenőrizve, hogy Dominik munkája összhangban legyen a sysfs újabb szolgáltatásaival. Valamivel később, de még ugyanebben a hónapban *Stanley Wang* kódot küldött *Greg Kroah-Hartman*-nak – ez a kód a sysfs-felülettel váltja fel a `pcihpfs`-t. Ebben az esetben azonban a sysfs még nem volt alkalmas a feladatra, mivel a szükséges, a gyorscserét lehetővé tevő kód még nem készült el teljesen. Sebaj, Greg kódolta a hiányzó sysfs-szolgáltatást, és elküldte Patricknak.
2003 januárjában egy szép napon *Alan Cox* megemlítette, hogy a tty kódja a 2.5-ös fában nagyon rosszul működik, és igazából már jó ideje olyan, amilyen, elsősorban azért, mert a rendszermag kódjában megváltozott a zárolás. Ez sokakat meglepett, és voltak, akik csodálkoztak, hogy miért csak most hallanak erről, amikor a 2.5-ös fa bővítését már befagyasztották, és a 2.6-os vagy a 3.0-s változat következik. Greg Kroah-Hartman megvizsgálta a hibát, és szörnyű dolgokat tapasztalt. Azt mondta, hogy a kódot nem lesz könnyű kijavítani, és valószínűleg csak a következő fa fejlesztéseként kerülhet rá sor. Alan erre azt válaszolta, hogy ezt nem tehetik, mivel a tty kódja rossz, és még a következő megbízható kiadás előtt ki kell javítani.
A Linux-rendszermag hagyományosan csakis a GNU C fordítóprogrammal működött együtt, és még így is gyakran előfordult, hogy a rendszermag egyes részeit csak a fordítóprogram egy bizonyos változatával lehetett lefordítani. A rendszermag mindig a GCC bővítéseire támaszkodott, s a mag és a fordítóprogram sorsa úgy egybefonódott, mint egy idős házaspáré. Így aztán sokakat megdöbbentett a hír, hogy a rendszermag fordításához az Intel C++-fordítóját, az icc-t is használni lehet. Úgy tűnik, az Intel már jó ideje célul tűzte ki, sőt javítófájlokat küldött Linusnak pusztán annak érdekében, hogy a fordítóprogramjuk kezelni tudja a rendszermag forrásfáját.

Zack Brown

Linux Journal 2003. május, 109. szám



A Linux Journal honlapján számtalan gond megoldásához találhattok további segítséget. A Sunsite tükröoldalait, a gyakori kérdéseket és az egyéb útmutatásokat a ➲www.linuxjournal.com honlapon olvashatjátok el. A rovatban közzétett válaszokat Linux-szakértők kis csapata készítette el. További kérdéseiteket szívesen fogadják (angol nyelven) a ➲www.linuxjournal.com/lj-issues/techsup.html címen, ahol csak egy kérdőívet kell kitöltenetek, de a bts@ssc.com címre levelet is írhattok. A levél tárgyában szerepeljen a „BTS" kulcsszó.

A hónap szakmai tanácsai

Az új Dell kiszolgáló lefagy

Munkahelyemen az első linuxos webkiszolgáló telepítésére készülünk, és ez örömmel tölt el. Gondjaim vannak viszont a kiszolgálókkal, remélem, tudtok segíteni. A fejlesztésre használt kiszolgálóink Dell 2550-es gépek, az éles kiszolgálók Dell 2650-esek. Red Hat 8.0-t futtatunk a vasakon, ami jórészt gond nélkül megy. Ugyanakkor megmagyarázhatatlan lefagyások történnek az összes kiszolgálón, amikor a konzol lefagy és a gépet újra kell indítani. A lefagyás oka nem jelenik meg a rendszernaplóban. A Dell és a Red Hat fórumain találtam némi segítséget. A lényeg annyi, hogy a *grub.conf* állományban a rendszermagnak át kell adni a `noapic` kapcsolót. Ezután a gépek szemmel láthatóan jól működnek. Mi a `noapic` kapcsoló hatása egy SMP-rendszeren? Tapasztalta más is ezt a hibát Dell 2550, illetve 2650-es gépeken?
Doug Farrell, dfarrell@grolier.com

Az APIC (fejlett programozható megszakításvezérlő) a szabványos, külső megszakításvezérlőt egy processzoron belüli megoldással váltja fel. Ez bizonyos ügyes trükköket tesz lehetővé, ilyenek például a teljesítményszámlálók és a figyelőszolgáltatások. Szokásos esetben ez a támogatás nem ütközik az olyan rendszerekkel, amelyekben nincs APIC. Bizonyos esetekben viszont az általad is tapasztalt lefagyásokhoz vezethet. A `noapic` módban történő futtatás legfőbb következménye a teljesítménycsökkenés, mert a megszakításokat a rendszer nem a leghatékonyabb módon kezeli. Az olyan rendszereknél, ahol sok a megszakítás (sajnos a nagy hálózati forgalmat bonyolító gépek – például a webkiszolgálók – ilyenek), ez a teljesítménycsökkenés észrevehető mértékű is lehet. Mindazonáltal az SMP előnyei gyakorlatilag mindig ellensúlyozzák ezt a hatást. Végezz terheléspróbát, hogy megtudd, mekkora a legnagyobb felhasználói terhelés, amit a rendszer még elvisel.
Chad Robinson, crobinson@rfgonline.com

Belkin vezeték nélküli kártya: támogatott?

Hordozható gépemről Belkin márkájú vezeték nélküli PCMCIA-kártyámmal a vezeték nélküli elérési ponthoz próbálok csatlakozni. Érdekelne, hogy milyen modult kell használnom a PCMCIA-kártyához.
Charles R. Fuller, charlesrfuller@netscape.net

Egy másik Linux-felhasználó volt olyan kedves, és megosztotta velünk a Belkin vezeték nélküli alkatrészeivel kapcsolatos tapasztalatait a honlapján. Ez a honlap neked is segíthet: ➲ http://www.jacked-in.org/linux/belkin_wireless.php.
Chad Robinson, crobinson@rfgonline.com

A Belkin kártyája ugyanazt a lapkakészletet használja, mint az Orinoco kártya. Egyszerűen a */etc/modules.conf* állományban *orinoco_cs* néven hozz létre egy álnevet a vezeték nélküli eszközhöz. Ha ez nem működik, a `cardctl ident` segítségével többet is megtudhatsz a lapkakészletről.
Christopher Wingert, cwingert@qualcomm.com

Hogyan lehet rendszerindításkor elindítani az XDM-et?

Nem tudom rávenni a Linuxot (Red Hat 7.2), hogy az X-es GUI-t indítsa el. Ehelyett szöveges bejelentkezési képernyőt kapok. Hogyan szerkeszthetem az alapértelmezett `init` szintet?
Keith Raposo, keith.raposo@sms.siemens.com

Ellenőrizd, hogy az X megfelelően van-e telepítve, ehhez írd be, hogy `startx`. Ha ez működik, akkor a */etc/inittab* állományban az első nem megjegyzéssort változtasd meg erre:
`id:5:initdefault:`
Usman S. Ansari, uansari@yahoo.com

A */etc/inittab* állományban van egy sor, ami `id:SZ`M:initdefault:` alakú. A számot változtasd a kívánt `init` szintre, ez az XDM bejelentkező képernyője esetén 5.
Christopher Wingert, cwingert@qualcomm.com

További segítség az SSH-val kapcsolatos kérdéshez

A 2003. áprilisi számban szerepelt egy kérdés, ami egyre gyakrabban előkerül, mert az emberek (és a terjesztések is) a nagyobb biztonságot választják alapértelmezettként vagy választható lehetőségként. Ha a felhasználó nem tud SSH-n keresztül csatlakozni, az a */etc/hosts.allow* és a */etc/hosts.deny* állományok miatt is lehet. A *hosts.allow* állományban állítsuk be az `sshd: ALL` értéket, vagy még jobb, ha ismerve azokat a gépeket, amelyekről ssh-zni kívánunk, csak ezeket soroljuk fel.
Benjamin Judson

Csatlakozás az MSN-hez

Modemem a KInternet programhoz van beállítva, ezt a programot használom a KDE munkaasztalon SuSE 8.0 alatt. A modem rendben elindul, hívja az internetszolgáltatóm (MSN) kiszolgálóját, majd ezt követően meghal. Az eseménynaplóban a következő hibaüzeneteket találtam:
`Failed Authentication with peer`
`Possible Bad Account or Bad Password`
Elképzelhető, hogy az MSN más bejelentkezési folyamatot kíván, mint ami a KInternet alatt be van állítva?
Chris, cgsnip@msn.com

Megpróbálhatsz más hitelesítési módokat is, ilyen például a PAP, illetve a CHAP.
Christopher Wingert, cwingert@qualcomm.com

Bizonyos felhasználók arról számoltak be a levelezési listákon, hogy sikeresen beléptek, miután az *MSN/* előtagot a felhasználónevük elé illesztették. Ha tehát a felhasználóneved joe, akkor a KInternetben az *MSN/joe* felhasználónevet állítsd be.
Don Marti, dmarti@ssc.com

Új termékek

S3C2500-RGP

Az Arcturus Networks és a Samsung Electronics közös munkájának eredményeként megszületett az S3C2500-RGP rendszer, ami otthoni átjárók, SOHO-hálózatok, internetre kötött eszközök és összevonó berendezések tervezéséhez szolgáltathat támpontot. A rendszer tartalmazza az alapgépet, amit modulokkal lehet kiegészíteni, a Linux operációs rendszert és választható gyári programkészleteket, amelyek bővítik a termék tudását és felgyorsítják a tervezést. A termék a Samsung ARM940 S3C2500 processzorára épül, és a μClinux működteti. A rendszer kiépítése: 4 MB Flash ROM, 8 MB SDRAM, egy 100BaseT ethernetkapu, négy 100BaseT ethernet LAN-kapcsoló, egy lapkába épített titkosítási gyorsító, két soros kapu, PCMCIA- támogatás és I2C soros EPROM. Lehetséges a WiFi, WiFi/WAP, többkapus DSP hang és SmartCardos VPN-hitelesítés támogatása is.

Adatok: Arcturus Networks, Inc., 116 Spadina Avenue, Suite 100, Toronto, Ontario M5V 2K6, Canada, telefon: 416-621-0125, e-mail: info@arcturusnetworks.com, ➲ http://www.arcturusnetworks.com

FAXCOM Server on Linux

A FAXCOM Server on Linux a menet közbeni dokumentumátalakítás használatával több eltérő csatolt dokumentum támogatására képes, és legfeljebb 96 kaput támogat a faxkiszolgálón. A beérkezett faxokat kiterjesztett faxközvetítő lehetőségek segítségével osztályozhatjuk, többek között a faxkapu, a tárcsázott faxszám, a küldő TSID-je és a hívóazonosító szerint. Ha szükséges, az adott faxot több célba is eljuttathatjuk, például több nyomtatóra.

A FAXCOM Server on Linux egy szűrőt tartalmaz a kéretlen faxok kiszűrésére, ami a bejövő faxok közül kiválogatja a szemetet, és egy kijelölt könyvtárba különíti el őket.

A FAXCOM Server on Linux tartalmazza a kötegelt faxfeladás lehetőségét is, ami lehetővé teszi, hogy a felhasználó egy hívással több faxot küldjön ugyanarra a telefonszámra.

Adatok: Biscom, Inc., 321 Billerica Road, Chelmsford, Massachusetts 01824, telefon: 800-477-2472, e-mail: sales@biscom.com, ➲ http:// www.biscom.com

SNAP Ultimate I/O Learning Center

A SNAP Ultimate I/O Learning Center egy olyan önálló rendszer, ami a felhasználók számára lehetővé teszi, hogy az Opto 22 SNAP Ultimate I/O rendszerével tanuljanak és gyakoroljanak. A Learning Center egy SNAP Ultimate processzort, válogatott B/K-modulokat, egy SNAP keretet, tápegységet, betöltőpanelt és vezetékeket, valamint a kézikönyveket és programokat tartalmaz, amelyek a felhasználókat segítik a valós idejű ipari automatizálási és befogási–szállítási alkalmazások elkészítésében. A Learning Center segítségével a gyakorlatban kipróbálható a B/K-kapuk beállítása, az egyszerű vezérlési stratégiák írása és grafikus felhasználói felület építése.

Adatok: Opto 22, Inc., 43044 Business Park Drive, Temecula, California 92590, 800-321-OPTO, ➲ http:// www.opto22.com

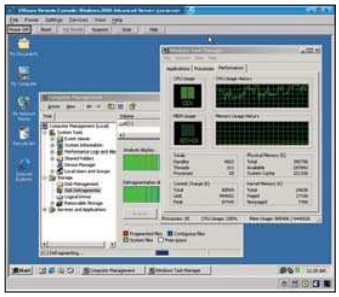
Quicknet VoIP Linuxra

A Quicknet Technologies bejelentette Linux Special Edition nevű termékét, ami a GnomeMeeting programmal és a Quicknet MicroTelco VoIP szolgáltatásával együtt telefonbeszélgetések kezdeményezésére és fogadására használható – az Interneten keresztül hagyományos telefonkészülékkel. A Special Edition termékben megtalálható az internetes Phone-JACK-PCI, LineJACK-ISA és Phone-CARD-PCMCIA bővítőkártya, ami a minőségi hangátvitelt biztosítja. A kártyákhoz való nyílt forrású illesztőprogramok a rendszermag részei, ez a Quicknet VoIP szolgáltatásával, az OpenH323 protokollokkal és a GnomeMeeting programmal kiegészítve olcsó, internetalapú telefonálást tesz lehetővé.

Adatok: Quicknet Technologies, Inc., 520 Townsend Street, San Francisco, California 94103, telefon: 415-864-5225, ➲ http:// www.quicknet.net

GSX Server 2.5

Megjelent a VMware GSX Server legújabb változata, a 2.5-ös. A nagyvállalati szintű virtuálisgép-programot adatközpontok üzleti szempontból lényeges alkalmazásaihoz és egyéb nagy forgalmat igénylő feladatokhoz tervezték.



A GSX Server biztonságos és egységes felületet biztosít a kiszolgálók összevonásához vagy felosztásához, hogy az erőforrások kihasználása és kezelése a leghatékonyabb lehessen. Több operációs rendszert és azok alkalmazásait is képes egy időben futtatni egy egyedülálló Intel alapú rendszeren. A 2.5-ös változat új képessége a legfeljebb 64 GB-os memória, továbbá 32 processzor és 64 virtuális gép támogatása.

Adatok: VMware, Inc., 3145 Porter Drive, Palo Alto, California 94304, telefon: 877-486-9273, e-mail: sales@vmware.com, ➲ http:// www.vmware.com

Mobile DiskOnChip G3

Az M-Systems bejelentette a 64 MB-os Mobile DiskOnChip G3 megjelentetését, amit a Toshibával közösen fejlesztettek ki sokoldalú mobileszközökhöz, például tenyérgépekhez vagy a 2.5G és 3G vezeték nélküli eszközökhöz. A G3 flashlemez többszintű cellán (MLC) és NAND Flash flashmemórián alapul, ami a szilíciumlapka méretét felére csökkenti, mivel cellánként nem egy, hanem két bitet tárol. A G3 az MLC NAND teljesítményszintjét a bináris NAND Flash értékei közelébe emeli. A G3 más kapcsolt és beágyazott eszközökhöz 7×10×1,2 mm FBGA (Fine-Pitch Ball Grid Array) tokozásban és TSOP-I alakban érhető el.

Adatok: M-Systems, Inc., 8371 Central Avenue, Suite A, Newark, California 94560, telefon: 510-494-2090, e-mail: info@m-sys.com, ➲ http:// www.m-sys.com

Linux Journal 2003, 109. szám

A jót tenni kell, a rosszat pedig megakadályozni

Helye van a rágalmak és a hátrányos jogi megkülönböztetések elleni küzdelemnek, de a Linuxszal végzett hasznos munkának is.

Harminc évvel ezelőtt a hanfordi nukleáris telepen dolgoztam. Munkakörömbe elsősorban a rendszerprogramozás tartozott, de végül rámragadt valami a környezetemből. Sokat tanultam az atomenergiáról és annak antiszociálisabb oldalairól, és felismertem néhány mélyen gyökerező gondot. Megszámlálhatatlan kérdés merült fel, az első három: a működési kockázat, a nukleáris hulladék kezelése és a teljes élettartamra vetített költség. Azt biztosan tudtam, hogy az atomenergia nem lesz – Eisenhower elnök húsz évvel korábbi szavaival élve – „olcsóbb annál, hogy mérni kelljen".

Azt akartam, hogy mindenki megtudja az atomenergiáról mindazt, amit én. De azt is szerettem volna, hogy mindenki megtudja, amit az olyan megújuló energiaforrásokról tudtam meg, mint a napenergia, a biomassza és a szélenergia. Hamar rájöttem, hogy nem foglalkozhatok mindkét üggyel. A megújuló energia megismerése és ezeknek az ismereteknek a tanítása nagy feladat, csakúgy, mint az atomenergia hibáinak a bemutatása. Tehát választottam. Mindazok után, amit Hanfordban megtanultam, úgy éreztem, jobban fel vagyok készülve arra, hogy az atomenergia előállításával kapcsolatos gondokról beszéljek.

Ez a döntés a „rossz megakadályozása" mellett szólt. Minél több ismeretet sikerül átadnom, annál nagyobb az esélye, hogy a többség megértse a helyzetet, szerepet vállaljon és – hosszabb távon – megakadályozza, hogy az Egyesült Államok és a világ még mélyebbre süllyedjen a nukleáris mocsárban.

Sajnos időközben újabb „rossz" jelent meg. Ahogy a napelemek egyre olcsóbbak lettek, az áramszolgáltatók azért kezdtek lobbizni, hogy a fogyasztóknak nehezebb legyen a hálózatra visszatáplált áramot eladni. Amikor bebizonyosodott, hogy a visszatápláló rendszerek biztonságosak és hatékonyak, az áramszolgáltatók az árakat úgy határozták meg, hogy a fogyasztók által visszatáplált áram olcsóbb legyen, mint amit ők adnak el a fogyasztóknak, noha a napelemes rendszerek a legnagyobb teljesítménnyel éppen csúcsidőben üzemelnek. A törvényt akarták használni a hibák következményeinek a kiküszöbölésére, úgy, hogy a visszatápláló rendszerekkel fizettették meg az atomerőművek veszélyeinek ártalmatlanítási költségeit.

Most térjünk vissza a programok világába. Amikor először találkoztam a Linuxszal, pusztán „valódi" Unix-rendszerek alternatíváját kerestem.

Úgy éreztem, hogy a Linux nagyon ígéretesnek mutatkozik, ezért irányt váltottunk, és unixos cégből linuxos céggé alakultunk. Most, hogy a Linux Journal több mint száz kiadását tudhatjuk magunk mögött, úgy érzem, jól határoztunk.

Kezdetben sok energiát fektettem abba, hogy a Linux erényeiről beszéljek az embereknek. Hasonlóan ahhoz a gondolathoz, hogy az atomenergia túl sokba kerül, kezdetben ezt is nehéz volt eladni. Az emberek nem voltak fogékonyak arra a gondolatra, hogy a Linux jobb választás lehet, mint amivel éppen dolgoztak.

Hol vannak már azok az idők, amikor arról kellett meggyőzni az embereket, hogy a Linux komoly kihívó az operációs rendszerek piacán! Mégha nem is Linux van a felhasználó gépén, nehezen képzelhető el, hogy webböngészés közben ne kerülne kapcsolatba valamilyen linuxos kiszolgálóval.

A gond az, hogy éppen úgy, ahogyan a háztetőn lévő napelempanelek fenyegetést jelentenek az atomenergia-üzletre, a Linux is fenyegetést jelent az operációs rendszerek status quójára nézve. Sok időt és energiát lehet a rágalmak elleni küzdelemre fordítani.

Ahogy a Linux egyre értékesebb választási lehetőséggé válik, ugyanolyan „működik, de itt nem használhatod" típusú érvek jelennek meg, mint az elektromos hálózatra visszatáplált választható energiával szemben. Éppen úgy, mint az atomenergia és a napenergia kapcsán, szép dolog a verseny, a tapasztalat azonban azt mutatja, hogy az ígéretes új lehetőségeket a jogrendszer segítségével hozzák hátrányba.

Ezt a „rossz megakadályozása" érdekében végzett munkát is el kell végezni. De van egy másik lehetőség: az, hogy egyszerűen csak haladunk a Linux útján – megkeressük a helyeket, ahol a Linux megold egy feladatot, és megvalósítjuk a megoldást. Olyan sok helyen alkalmaznak időrabló, nem önműködő megoldásokat vagy gyengén kivitelezett nem-linuxos rendszereket, hogy könnyű érvényesülni a linuxos feladatmegoldással.

A Linux-mozgalomnak mindkettőre szüksége van. Valakinek foglalkoznia kell a rágalmakkal, és valakinek új helyekre kell eljuttatnia a Linuxot.

Ami engem illet, kivettem a részemet a rágalmak elleni küzdelemből. Legtöbbször ez szórakoztató munka volt, de mára szinte teljesen az „egyszerűen csak oldjuk meg Linuxszal" táborba álltam át. Inkább megmutatom a megoldást, és felkínálom a döntési lehetőséget valakinek, minthogy a linuxellenes propaganda leküzdésére pazaroljam az időmet.

Több mint egy éve Costa Ricába költöztem. A legnagyobb különbség, amit ott tapasztaltam, az az volt, hogy az emberek itt nyitottabbak a megoldásokra. Itt kevesebb az elköltésre váró pénz, mint az Államokban, és kevesebb a Linux elleni rágalom is. Ezért itt könnyebben megtörténhet, hogy meghallgatom a gondot, Linux alapú megoldást ajánlok, és azt el is fogadják. Könnyebben hiszi el az ember, amit Linus a világuralomról mondott – csak lehet, hogy az USA lesz az utolsó ország, amelyik kiaknázza a Linuxban rejlő lehetőségeket. Egyébként amellett, hogy Costa Rica Linux-barát, az ország teljes elektromos ellátását megújuló energiaforrások szolgáltatják, beleértve a víz-, a geotermikus, a szél- és a napenergiát. Ez a két ügy talán szorosabban összefügg, mint gondoltam.

Linux Journal 2003. május, 109. szám

Phil Hughes
A Linux Journal kiadója.

Ismerjük meg a 2.6-os rendszermagot!

Az új ütemező és hangalrendszer mindössze kettő a felfedezendő képességek közül, ami csak ránk vár, amint a 2.5-ös fejlesztői változat végre 2.6-ra vált. Hallgassuk meg, mit gondol erről egy magprogramozó!

A rendszermag hosszú fejlődésen ment keresztül, amióta Linus 2001. november 22-én a 2.4.15 változatból kiemelve létrehozta a 2.5.0-t. Ezután beindult a fejlesztés, aminek eredményeképpen egy teljesen más, jóval többre képes rendszermag jött létre. Ebben a cikkben az érdekesebb, fontosabb képességeket emeljük ki, illetve megnézzük, hogy ezek milyen hatással vannak a Linux teljesítményére és megbízhatóságára.

A 2.5-ös változat története napjainkig

A linuxos hagyományoknak megfelelően a rendszermag kisebbik változatszámából mindig kiolvashatjuk, hogy az adott rendszermag a fejlesztői vagy az üzembiztos sorozatba tartozik-e. Az üzembiztos rendszermagokat páros kis változatszámok jelölik, a fejlesztői változatok pedig páratlan változatszámot kapnak. Amikor már a fejlesztői változat teljesen kiforrott, és megbízhatónak minősítik, kis változatszámát a következő páros számra növelik. Például, a 2.4-es megbízható magsorozat a 2.3-as fejlesztői rendszermagból született.
A jelenlegi fejlesztői mag változatszáma 2.5. A fejlesztői sorozat kezdeti munkái általában igen élénken indulnak, és számos új képesség és fejlesztés kerül bele ilyenkor a rendszermagba. Ha Linus és a magfejlesztők elégedettek az új képességkészlettel, bejelentik a képességbefagyasztást (feature-freeze), ennek célja a fejlesztés ütemének lassítása. Az utolsó képességbefagyasztás 2002. október 31-én volt. Eszményi esetben a képességbefagyasztást követően Linus már nem fogad el újabb képességeket – kizárólag a már meglévő munkarészek bővítését. Amikor a kiválasztott képességek elkészültek és csaknem teljesen stabilak, kihirdetik a kódbefagyasztást (code-freeze). A kódbefagyasztás ideje alatt kizárólag a hibajavításokat fogadják el, hogy a rendszermag végre megérhesse a megbízható kiadást.
Amikor a fejlesztői kiadás elkészül, Linus bejelenti az üzembiztos változatot. A jelenlegi üzembiztos rendszermag nagy valószínűséggel a 2.6.0 változatnevet viseli majd. Bár a hivatalos kiadás időpontját akkor tudjuk meg, „amikor elkészült" a rendszermag, 2003 harmadik vagy negyedik negyede viszonylag jó becslésnek tűnik.
2001 márciusában, majd 2002 júniusában a rendszermag vezető fejlesztői a Kernel Summits keretében találkoztak és vitatták meg a feladatokat. A 2.5-ös elsődleges célja az elöregedő blokkrendszernek (a rendszermag e része felelős a blokkeszközökért, például a merevlemezért) a XXI. századi követelményeknek megfelelő korszerűsítése volt. A további célok közt találjuk a méretezhetőséget, a válaszidő és a virtuális memória (VM) fejlesztését. A rendszermagírók el is érték valamennyi felsorolt – és sok egyéb – célt. Az alábbiakban felsoroltuk a fontosabb új képességeket:

- O(1) ütemező,
- időosztásos rendszermag,
- lappangási fejlesztések,
- újratervezett blokkréteg,
- jobb VM alrendszer,
- jobb száltámogatás,
- új hangréteg.

Ebben a cikkben számos új módszerről és tervezetről fogok majd beszélni, amelyek bekerültek a 2.5-ös rendszermagba, és a 2.6-ban jelennek majd meg. A fejlesztés sok ember kemény munkájának az eredménye. Tartózkodni fogok a nevektől, hiszen ha elkezdenék köszöntet nyilvánítani, elkerülhetetlenül megfeledkeznék néhány emberről, így inkább semmilyen listát sem írok, mintsem hiányos, netán hibás felsorolást adjak közre. Ha kíváncsiak vagyunk rá, hogy melyik rész kinek a műve, a Linux Kernel Mailing List archívuma nagyon jó kiindulási alapot kínál hozzá.

O(1) ütemező

A folyamatütemező (vagy egyszerűen csak ütemező) a rendszermagnak az az alrendszere, amelyik a processzoridő foglalásáért felelős. Ez a rész dönti el, hogy mikor melyik folyamat fog futni. Ez korántsem mindig olyan egyszerű: az ütemezőnek akár egy hosszú listából is ki kell tudnia választani a futtatandó legértékesebb folyamatot. Amikor nagy számú futtatható folyamatunk van, a legjobb folyamat kiválasztása időbe telhet; a többprocesszoros gépek pedig további kihívást jelentenek.
A szükséges módosítások listáján az ütemező igen előkelő helyet foglalt el. A fejlesztőknek három fő céljuk volt, részletesebben:

- Az ütemezőnek teljes körű O(1) ütemezést kell nyújtania. Az ütemező minden algoritmusa állandó idő alatt végezzen, a futó folyamatok számától függetlenül.
- Az ütemezőnek kifogástalan SMP-méretezhetőséggel kell rendelkeznie. Eszményi esetben minden egyes processzornak külön zárolási rendszere és futtatási sora lenne. A futtatási sor a folyamatoknak az a listája, amiből az ütemező választhat.
- Az ütemezőnek jobb SMP-képességekre van szüksége. Képesnek kell lennie csoportosítani folyamatokat az egyik központi egységre, és ott lefuttatni őket. A futtatási sorok hosszának kiegyensúlyozatlanságait feloldandó lehetőséget kell adni, hogy a folyamatok az egyik processzorról a másikra kerülhessenek.

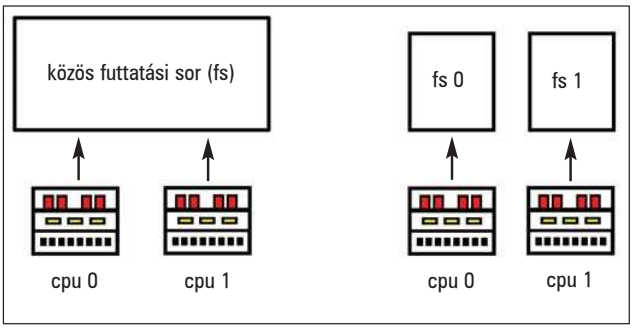
Az új ütemező valamennyi fenti célt megvalósította. Az első cél a teljes O(1) ütemezés volt. Az O(1) olyan algoritmust jelöl, ami állandó (konstans) idő alatt hajtódik végre. A rendszeren futó folyamatok száma – vagy bármilyen más változó – nincs hatással az ütemező egyik részének végrehajtási idejére sem. Képzeljünk el egy algoritmust, amelyik eldönti, hogy melyik folyamat fog következő lépésben futni. Meg kell találunk a legnagyobb prioritású futtatható folyamatot, ami még maradék időszelettel rendelkezik. A korábbi ütemezőben az algoritmus szerkezete a következő volt:

```
for (minden futtathat  folyamatra) {
        keresd a folyamat ørtØkessØgØt
        if (ha ez a legØrtØkesebb folyamat
eddig) {
                emlØkezz rÆ
        }
}
futtasd a legØrtØkesebb folyamatot
```

A fenti algoritmus alapján minden egyes folyamat értékét le kell ellenőriznünk. Ennek megfelelően az algoritmus n elem esetén n-szer ismétel. Így O(n) algoritmusnak nevezzük, hiszen végrehajtási ideje a folyamatok számának növekedésével lineárisan növekszik.
Ezzel szemben az új ütemező a folyamatok számát nézve állandó végrehajtási idejű; azaz teljesen mindegy számára, hogy öt vagy 5000 futtatható folyamat van a rendszeren. Mindig pontosan ugyanannyi időre van szüksége a folyamat kiválasztására és végrehajtására:

- Kérd le a legmagasabb prioritást, amihez folyamatok tartoznak.
- Kérd le az adott prioritási szint listájának az első tagját.
- Futtasd ezt a folyamatot.

Ezzel az algoritmussal megoldhatók a „kérd le a legmagasabb prioritást” és a „kérd le a lista első tagját” műveletek, mivel ezeket az értékeket az ütemező folyamatosan nyilvántartja – keresés helyett egyszerűen csak le kell kérdeznie őket. Eredményképpen az új ütemező a folyamatok végignézése nélkül is képes a legmagasabb prioritású folyamatot kiválasztani.
A második cél a tökéletes SMP-méretezhetőség elérése volt. Ide tartozik, hogy az ütemező teljesítménye egy adott processzoron akkor is azonos marad, ha újabb processzorokat adunk a rendszerhez. A korábbi ütemezőben ez nem így volt: az ütemező teljesítménye a zárolási verseny következtében a folyamatok számának növekedésével egyre csökkent. Az ütemezőnek és adatszerkezeteinek a karbantartása meglehetősen költséges, s ennek legnagyobb részét a futtatási sor kezelése tette ki. Egy zárolásnak nevezett műveletet használunk annak érdekében, hogy egyszerre csak egy processzor tudjon változtatni a futtatási soron. Ez lényegében azt jelenti, hogy az ütemezőt egyszerre csak egy processzor futtathatja.
Ezt a problémát orvosolandó az új ütemező egyetlen közös futtatási sor helyett minden processzorhoz külön sort használ. Ezt a megvalósítást gyakran többsoros ütemezőnek is nevezik. Minden egyes processzor futtatási sorában külön folyamatlista várakozik a végrehajtásra. Amikor egy adott processzor végrehajtja az ütemezőt, csak a saját futtatási sorából választ. Ennek következtében a futtatási sorban sokkal kisebb lesz a versengés, és a teljesítmény a rendszer processzorszámának növekedésével nem csökken. Az *1. ábrán* láthatjuk a közös futtatási sorral és egyedi futtatási sorokkal rendelkező kétprocesszoros gépet.
A harmadik és egyben utolsó cél a fejlettebb SMP-affinitás elérése volt. A korábbi Linux-ütemező rendelkezett egy nemkívánatos jellemzővel, nevezetesen: a folyamatokat pattogtatta a processzorok között. A fejlesztők ezt a viselkedést pingpong-hatásnak nevezték el. Az *1. táblázat* a legrosszabb ilyen esetet mutatja be.
Az új ütemező a processzoronkénti külön futtatási soroknak köszönhetően képes feloldani ezt a nehézséget. Mivel minden processzor saját listával rendelkezik a futtatható folyamatokról, a folyamatok azonos processzoron maradnak. A *2. táblázat* ezt a fejlettebb működési elvet mutatja be. Természetesen néha



1. ábra Bal oldalt a 2.4-es futtatási sora; jobb oldalon a 2.5-ös, illetve a 2.6-os futtatási sora

előfordul, hogy a folyamatokat át kell tenni egyik processzorról a másikra, például ha a processzorokon futtatott folyamatok száma nincs egyensúlyban. Ilyen esetekben a különleges, terhelés-egyensúlyozó módszer kiszedheti a folyamatokat a sorokból. Ez a művelet viszonylag ritkán fordul elő, így az SMP-affinitás megmarad.
Az új ütemező sokkal többre képes, mint gondolnánk. A *3. táblázatban* az új ütemező teljesítménypróbáit adjuk közre.

Időosztásos rendszermag

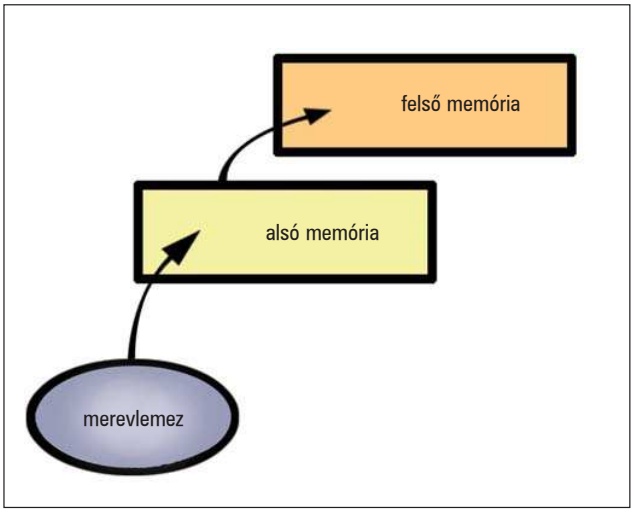
A rendszermag-időosztás feladata a lappangási idő csökkentése. Az eredmény: gyorsabb rendszerválaszidő és kellemesebb interaktív élmény lesz. A Linux-rendszermag a 2.5.4 változattól kezdve vált időosztásossá (preemptív), korábban együttműködő módon hajtódott végre. Ez annyit jelent, hogy a folyamatok (legyenek akár valós idejűek) a rendszermag rendszerhívásaiban nem tudtak más folyamatokat előreengedni.
Ennek eredményeképpen az alacsony prioritású folyamatok a magasabb rendűek elé kerülhetnek, megakadályozva, hogy azok igényeik szerint érjék el a processzort. Mégha az alacsonyabb rendű folyamat időszelete le is járt, tovább folytathatja a futását, amíg a rendszermagon belül be nem fejezte az elkezdett feladatot, vagy önként vissza nem adja az irányítást. Ha a várakozó magasabb rendű folyamat például egy szövegszerkesztő, amibe a felhasználó éppen gépelni szeretne, vagy MP3-lejátszó, ami a hangvermét újra szeretné tölteni, az eredmény gyenge interaktív teljesítmény lesz. Ha viszont a magasabb rendű folyamat valamilyen különleges valós idejű folyamat, az eredmény egyenesen végzetes is lehet.
De miért nem volt már a kezdetektől fogva időosztásos a rendszermag? Azért, mert időosztásos rendszermagot készíteni sokkal több munkát jelent. Ha a rendszermagban futó felada-

1. táblázat **A pingponghatás legrosszabb esete**

CPU	idő 1	idő 2	idő 3	idő 4	idő 5
A folyamat	CPU 0	CPU 1	CPU 0	CPU 1	CPU 0
B folyamat	CPU 1	CPU 0	CPU 1	CPU 0	CPU 1

2. táblázat **Az új ütemező megőrzi a processzorirányultságot**

CPU	idő 1	idő 2	idő 3	idő 4	idő 5
A folyamat	CPU 0	CPU 0	CPU 0	CPU 0	CPU 0
B folyamat	CPU 1	CPU 1	CPU 1	CPU 1	CPU 1



2. ábra 2.4-es rendszermag pattintóveremmel

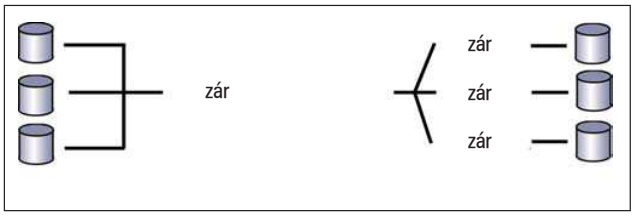
tok bármikor újraidőzíthetőek, védelmet kell beépíteni, hogy megakadályozzuk az osztott adatok egyidejű elérését. Szerencsére az időosztásos rendszermag gondjai éppen azonosak az SMP (symmetrical multiprocessing) esetében korábban felmerült gondokkal. Így aztán azt a módszert, ami eredetileg SMP alatt nyújtott védelmet, a rendszermag időosztási gondjainak megoldásához is könnyedén át lehetett ültetni. A rendszermag egyszerűen az SMP forgózárait (spinlocks) használja az időosztás jelzésére. Amikor a kódban zárolásra van szükség, az időosztás szintén kikapcsolódik. Minden más esetben a jelenlegi folyamat megelőzése teljesen biztonságos.

A lappangási idővel kapcsolatos fejlesztések

Valószínűleg már látjuk is a következő szűk keresztmetszetet. Az időosztásos rendszermag az időzítés lappangási idejét a rendszermag végrehajtási idejéről a forgózár végrehajtási idejére csökkenti. Ez egyértelműen gyorsabb, de még mindig lehetséges hibaforrás. Szerencsére a zárolási idő, vagyis az az időszak, amíg a rendszermag időosztásos módja ki van kapcsolva, csökkenthető.
A rendszermagfejlesztők az alacsony lappangási időt szem előtt tartva hatékonnyá tették a rendszermag algoritmusait. Elsősorban a VM és a VFS (Virtual FileSystem) feladataira összpontosítottak, ennek eredményképpen a zárolás ideje jelentősen csökkent. Végeredményül kitűnő válaszidőt kaptak. A felhasználók a 2.5-ös alatt még közepes gépeken sem tapasztaltak 500 nanosecundumnál rosszabb időzítési lappangási időt.

Újratervezett blokkréteg

A blokkréteg a rendszermagnak az a része, ami a blokkos eszközök kezeléséért felelős. A hagyományos Unix-eszközök kétfajta alkatrészt támogatnak: a karakteres és a blokkos eszközöket. A karakteres eszközök, például a soros kapuk és a billentyűzet, egyesével karakterek vagy bájtok folyamaként értelmezik az adatokat. Ezzel szemben a blokkos eszközök előre megadott méretű adagonként kezelik az adatokat (ezeket nevezzük blokknak). A blokkos eszközök nem csupán fogadják és küldik az adatfolyamokat; bármikor bármelyik blokkjuk elérhető. Az egyik blokkról a másikra történő ugrást keresésnek hívják (seeking). Blokkos eszközre példa a CD-ROM-meghajtó, a merevlemez és a szalagos egység.
A blokkos eszközök kezelése egyáltalán nem magától értetődő. A merevlemezek meglehetősen összetett alkatrészek, amelyek bármelyik érvényes blokkjához az operációs rendszernek írási és olvasási hozzáférést kell biztosítania. Mivel a keresés költséges, az elérési idő mérséklése érdekében az operációs rendszernek okosan kell kezelnie valamint besorolnia a blokkos eszközkéréseket.
A Linux blokkrétegére már nagyon ráfért az újratervezés. Szerencsére a 2.5.1-gyel megkezdődött a réteg kipofozása.
A munkába bevont legérdekesebb részek a blokkos I/O-kérelmeknek megfeleltetett új, rugalmas és általános szerkezet bevezetése, a pattintóvermek (bounce buffers) eltüntetése, és az I/O-folyamatoknak közvetlenül a magas memóriában történő támogatása, valamint a várakozási soronként létrehozott `io_request_lock` és az új I/O-ütemező.
A 2.5-ös változat előtt a blokkos eszköz az I/O-kérelmek leírására a `buffer_head` szerkezetet használta. Ez a módszer több okból sem volt hatékony, de a legkomolyabb érv ezek közül az, hogy a blokkos rétegnek az adatszerkezetet gyakran kisebb darabokra kellett törnie, majd az I/O-ütemezőben később újra össze kellett állítania. A 2.5-ös rendszermag az I/O-műveletek tárolására új adatszerkezetet használ, a bioszerkezetet (bio struct). Ez a szerkezet egyszerűbb, a nyers és átmeneti „tárazott" I/O-műveletekhez egyaránt felhasználható, a magas memóriában is működik, ráadásul könnyedén szétszedhető, illetve összerakható. A blokkos réteg egységesen az új bioszerkezetet használja, így a kód tisztább és hatékonyabb lesz.
A következő feladat a magas memóriában végzett I/O-műveletekhez használt pattintóverem kiiktatása volt. A 2.4-es rendszermagban minden, a blokkos eszközökről a magas memóriába irányuló I/O-átvitel kényszermegállót volt kénytelen beiktatni. A magas memória olyan, nem állandóan belapolt memóriarész, amihez a rendszermagnak különleges támogatást kell nyújtania. Az Intel x86 gépein az összes 1 GB feletti rész ilyen típusú. A magas memóriába irányuló minden I/O-kérelem (például a merevlemezről beolvasni egy fájlt valamely 1 GB feletti címre) kénytelen az alacsony memóriában található különleges pattintóvermet használni. A gyakorlat azt mutatja, hogy néhány eszköz képtelen megérteni a magas memóriacímeket. Ennek megfelelően az eszközöknek mindig alacsony címen kell az I/O-átvitelt elvégezniük. Amennyiben a célterület valójában a magas memóriában található, a blokkos eszköz adatait csak az alacsony memórián keresztül, „pattintva" lehet a magas memóriába juttatni (lásd a *2. ábrát*). Ez a további másolás sok felesleges munkát jelent. A 2.5-ös rendszermag már önműködően támogatja a közvetlen magas memóriaátvitelt, és az erre képes eszközök esetében megszünteti a pattintóverem-logikát.
A következő szűk keresztmetszet, amivel a fejlesztők szembetalálkoztak, a teljes körű I/O-kérelemzár volt. Minden blokkos eszközhöz egy kérelemsor tartozik, ami a blokkok I/O-kérelmeit, azaz az egyes blokkok írását és olvasását jelző egyedi bioszerkezeteket tárolja. A rendszermag – ahogy a meghajtók felveszik vagy törlik a kérelmeket – folyamatosan frissíti a sorokat. Az egyidejű módosítástól a sorokat az `io_request_lock` védelmezi – a kód csak akkor módosíthatja a sort, ha birtokolja a zárat. A 2.5 előtti rendszermagokban egyetlen teljes körű zár oltalmazta a rendszer összes kérelmét. A teljes körű zár az összes sor egyidejű elérését megakadályozta, holott a zárnak pusztán az egyetlen soron belüli egyidejű eléréseket kellene megakadályoznia. A 2.5-ös alatt a teljes körű zárat külön sorokhoz rendelt finom felbontású zárrendszerrel helyettesítették (*3. ábra*). Ennek megfelelően a rendszermag egy időben több sort is képes kezelni.



3. ábra A 2.5-ös rendszermagban megjelenik a kérelemsoronkénti külön zár

4. ábra A fordított térképezés egy fizikai lapot egy vagy több virtuális laphoz rendelhet

Végül az új I/O-ütemező kiküszöbölte a blokkos réteg maradék, kevésbé hatékony megoldását is. Az I/O-ütemező felelős a blokkok összerakásáért és fizikai eszközre történő küldéséért. Mivel a keresés költséges, az I/O-ütemező jobban szeret folyamatos kérelmeket teljesíteni, ezért a beérkezett kérelmeket szektorok szerint rendezi. Ez a képesség a lemezek teljesítménye és élettartama szempontjából egyaránt hasznos. A gond csak az, hogy a folyamatos szektorokra irányuló ismételt I/O-kérelmek megakadályozhatják egy nem ide tartozó szektor kérelmének a kiszolgálását. Az új I/O-ütemező úgy oldja meg ezt a nehézséget, hogy határidőt vezet be az I/O-kérelmekhez. Az I/O-ütemező csak a határidő lejártáig várakoztathatja a kérelmet, azután már mindenképpen ki kell szolgálnia, és nem folytathatja a jelenlegi szektorhoz tartozó kérelmek összeállítását. Az új ütemező egyben az írás–várakozás–olvasás kérdést is megoldja azáltal, hogy az olvasási műveleteknek előnyt ad az írással szemben. Ez a változtatás nagymértékben javítja az olvasási lappangási időt. Végül, de nem utolsó sorban, a kérelemsor mostantól egyszerű lista helyett vörös, illetve fekete fa alakú, ami igen könnyen kereshető adatszerkezet.

Fejlettebb VM alrendszer

A 2.5-ös változat alatt a VM végre magára talált. A VM alrendszer a rendszermagnak az a része, amelyik az összes folyamat virtuális címterületéért felelős. Ide értendő a memóriakezelési módszer, a lapkilakoltatási stratégia (mit cseréljünk le, ha kevés a memória) és a belapolási stratégia (mikor cseréljük vissza a dolgokat). A VM gyakran okozott nehézségeket Linux alatt. A bizonyos terhelés alatt mért jó VM-teljesítmény gyakran okoz gyenge teljesítményt más részeken. Az igazságos, egyszerű, jól felépített VM mindig is elérhetetlennek látszott – egészen mostanáig. Az új VM-ben végzett három nagyobb változtatás eredménye:

- fordított térképezéses (reverse-mapping avagy rmap) VM;
- újratervezett, okosabb, egyszerűbb algoritmusok;
- szorosabb együttműködés a VFS-réteggel.

A végeredményül kapott VM általános esetekben különlegesen jó teljesítményt nyújt, de szélsőséges esetekben sem hullik darabjaira. Lássuk mindhárom változást!
Minden virtuálismemória-rendszer rendelkezik fizikai címekkel (a fizikai RAM-lapkák tényleges lapcímeivel) és virtuális címekkel (az alkalmazások számára nyújtott logikai címekkel). A memóriakezelési egységekből (Memory Management Unit, azaz MMU-ból) álló szerkezetekkel kényelmesen kikereshetjük a virtuális címekhez tartozó fizikai címeket. Ez kívánatos is, mivel a programok folyamatosan használják a virtuális memóriacímeket, és ezt az eszköznek kell fizikai címmé alakítania. Ugyanakkor fordított irányban haladni egyáltalán nem olyan könnyű. Amennyiben fizikai címből szeretnénk virtuális címet létrehozni, a rendszermagnak minden táblabejegyzést végig kell néznie és ki kell keresnie a kívánt címet, de ez meglehetősen időigényes. A fordított térképezéses VM virtuális címekből fizikai címekre mutató térképeket is tartalmaz.

```
for (minden tÆblabejgyzØsre)
      if (ez a fizikai c m sz ksØges)
            megtalÆltuk a megfelelı c met
```

az `rmap` VM egy mutatót követve egyszerűen kikeresheti a virtuális címet. Ez a módszer sokkal gyorsabb, különösen nagyobb VM-terhelés alatt. A *4. ábra* a fordított térképezés diagramját ábrázolja.
Ezenkívül a VM programozói a VM több algoritmusát újratervezték és továbbfejlesztették, szem előtt tartva az egyszerűsítést, az általános esetekben elérendő nagy teljesítményt és a szélsőséges körülmények között szükséges legalább elfogadható teljesítményt. Az eredményül kapott VM egyszerűbb, mégis szívósabb lett.

3. táblázat **A csevegőkiszolgáló-teljesítménypróba nagyszámú folyamat közötti üzenetváltást mér. Az eredmények üzenet/másodperc mértékegységben értendők**

Ütemező	1. futás	2. futás	3. futás
2.4 ütemezője	79723	82210	94803
2.5 ütemezője	612320	620880	609420

4. táblázat **A szálkészítés és kilépés próbaeredménye: a teszt tíz kezdeti szál teljesítményét méri, amelyek mindegyike egy, öt vagy tíz párhuzamos szálat készít és zár be**

pthread könyvtár	1	5	10
LinuxThreads	140 µs	150 µs	170 µs
NGPT	75 µs	80 µs	90 µs
NPTL	20 µs	20 µs	20 µs

Végül sokat fejlődött a VM és a VFS közötti együttműködés. Ez létfontosságú, mivel a két alrendszer igen bensőséges viszonyban áll egymással. Egyszerűsödtek a fájl- és lapvisszaírások, az előreolvasás és a gyorstárkezelés. A `bdflush` rendszermagszálat a `pdflush` szálcsoport váltotta fel. Az új szálak sokkal jobb lemeztelítettség elérésére képesek; egy fejlesztő szerint a kód egy időben hatvan lemezfolyamatot tud telített állapotban tartani.

Szálak fejlesztése

Linux alatt a szálkezelés mindig utólag beleszőtt gondolatnak tűnt. A szálasított modell nem igazán illeszkedik a hagyományos Unix-folyamatmodellbe, ennek megfelelően a Linux-

rendszermagról sem mondható el, hogy elkényeztetné a szálakat. A felhasználói térben futó ***pthread*** programkönyvtár (közismertebb nevén a LinuxThreads), ami a glibc (a GNU C könyvtár) része, nemigen kap nagy segítséget a rendszermagtól, ennek eredményképpen a szálak teljesítménye sem éppen csillagászati. Rengeteg lehetőség kínálkozott a fejlesztésre, de ezt csak úgy lehetett kivitelezni, ha a rendszermag és a glibc programozói együtt tudnak működni.
Örvendezzünk, mert képesek voltak erre, következésképpen a rendszermag száltámogatási képessége jelentős mértékben megnőtt; és Native Posix Threading Library (NPTL) néven létrejött egy új, felhasználói térben használható ***pthread*** könyvtár, ami majd a LinuxThreads helyére léphet. Az NPTL, akárcsak a LinuxThreads, 1:1 arányú szálasító modellt használ. Ez annyit jelent, hogy minden felhasználói térben létező szálhoz pontosan egy rendszermagszál tartozik. Lenyűgöző, hogy a fejlesztők anélkül voltak képesek kitűnő teljesítményt elérni, hogy átálltak volna az M:N modellre (ahol rendszerszálak száma dinamikusan kevesebb is lehet, mint a felhasználói térben futó szálak száma). A magváltozások és az NPTL együtt növekvő teljesítményt és szabályosságot hozott. Egy kis ízelítő az újdonságokból:

- a szálak helyi tárolástámogatása,
- O(1) `exit()` rendszerhívás,
- fejlettebb PID-foglaló rendszer,
- a `clone()` rendszerhívás szálasító bővítése,
- szálérzékeny kódkiíró- (dump) támogatás,
- szálasított jelzés- (signal) fejlesztések,
- új, gyors, felhasználói térben futó zárolási primitívek (ezeket futexnek nevezzük).

Az eredmény önmagáért beszél. Egy adott gépen 2.5-ös rendszermag és az NPTL alkalmazásával százezer szál egyidejű létrehozása és törlése kevesebb mint két másodpercig tart. Ugyanitt a rendszermag-változtatások és az NPTL nélkül ugyanez a próba körülbelül 15 percet vesz igénybe.
A 4. ábrán láthatjuk a szálkészítő és szálkilépő próbateljesítmény eredményeit, az NPTL, NGPT (az IBM M:N arányú Next Generation POSIX Threads nevű ***pthread*** könyvtára) és a LinuxThreads rendszerek felhasználásával. Ez a próba szintén százezer szálat készít, de sokkal kisebb párhuzamos lépésekkel. Ha még ez sem nyűgözött le, akkor igazán keményfejű vagy.

Új hangréteg

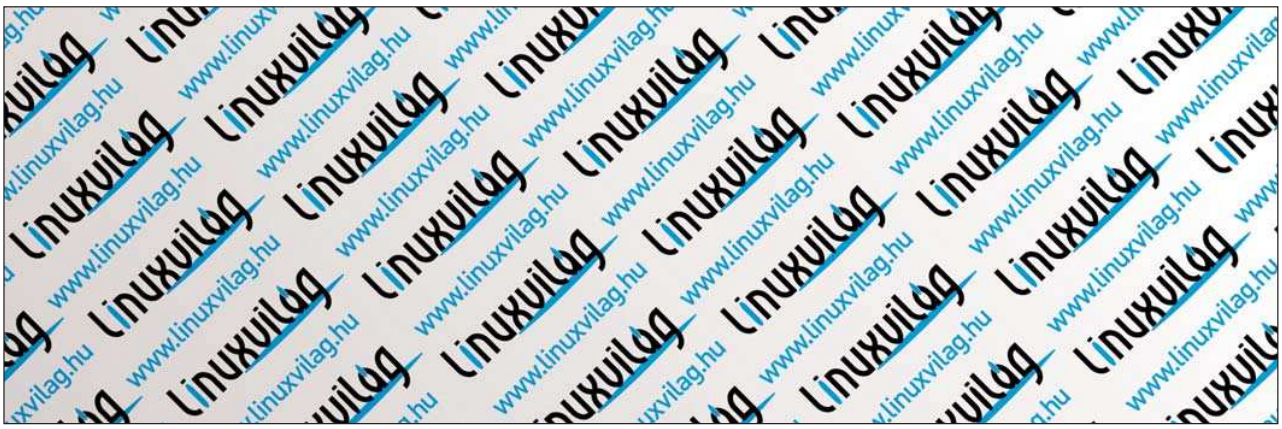
A Linux sound architecture (ALSA) régóta várt beillesztése a 2.5.5-ös változatban kezdődött meg. Az ALSA rengeteg fejlesztést tartalmaz az előző hangréteg, az Open Sound Sytem (OSS) rendszerhez képest. Mind közül a legfontosabb, hogy az ALSA sokkal erősebb és gazdagabb API-t nyújt, mint az OSS. Az ALSA-meghajtók és a hozzájuk tartozó programkönyvtár (alsa-lib) segítségével kevesebb erőfeszítéssel tudunk fejlett audioalkalmazásokat készíteni.
Az ALSA rengeteg hangeszközt támogat, továbbá visszafelé együttműködő OSS-csatolófelülete is létezik. Nagyon valószínű azonban, hogy az OSS-meghajtók a 2.6-osban is megmaradnak azok kedvéért, akiknek továbbra is OSS-re van szükségük, esetleg jobban kedvelik.

Jövőkép

Egy kicsit talán felelőtlen dolog a 2.6-os jövőjét boncolgatni, hiszen még meg sem jelent. Ugyanakkor érdekes elgondolkodni azon, milyen is lesz a 2.7-es fejlesztői rendszermag (vagy legalábbis milyet szeretnénk). Ha szerencsénk van, most a régóta várt tty-réteg (terminál) újraírása következik. A tty-réteg mostanra ugyanis hatalmasra duzzadt, és eléggé zavarossá vált.
Mindenki kívánságlistáján szintén az első helyezettek között áll a SCSI-réteg újraírása. Jelenleg a SCSI-réteg túlságosan buta, a meghajtók pedig túl okosak. Sőt az IDE- és SCSI-rétegeket is egyesíteni lehetne, akár egyetlen általános lemezrétegben! De akárhogy is legyen, a SCSI-rétegnek mindenképpen szüksége van egy kis tisztogatásra.
A fentieken kívül minden más elég bizonytalan. Kockázatos lenne találgatásokba bocsátkozni; az eddig felsoroltak is inkább csak a jelenlegi igények egyszerű megfigyelése alapján születtek. Mint mindig, a 2.7-es tényleges munkái is – akárcsak a vakaródzás – attól függnek majd, hogy hol viszket a fejlesztőknek.
A jövőtől függetlenül a 2.6-os rendszermag nagyszerűnek tűnik – kiváló méretezhetőség, fürge munkafelületi válaszidő, fejlettebb igazságosság, illetve boldogan együttműködő VM és VFS-réteg jellemzi.

Linux Journal 2003. május, 109. szám

Robert Love (rml@tech9.net)
Matematika és számítógépes tudományok szakos hallgató a Floridai Egyetemen. Amikor éppen nem Linuxot elemez, autóversenyzik, thai ételeket eszik vagy punk zenét hallgat.



A rendszermag beállításai és fordítása

A 2.5-ös sorozat esetében a korábbinál sokkal egyszerűbb saját rendszermagot készíteni vagy illesztőprogrammal bővíteni a rendszerünket.

A rendszermagfordítás művelete két részből áll: először meg kell adni a megfelelő beállításokat, majd ezekkel a beállításokkal el kell végezni magát a fordítást. A 2.5-ös előtti magváltozatoknál a beállítások a ***Config.in*** nevű állományokba kerültek, ilyen minden alkönyvtárban volt; súgóként pedig egy központi leírás, a ***Documentation/Configure.help*** fájl szolgált. A fordítási folyamat leírására szolgáló nyelv parancssori jelleget öltött, és a felhasználónak felkínálandó beállításokat az éppen megjelenített beállításoktól függően választotta ki.
A dolog viszonylag jól működött, ám idővel túl sok különféle beállítás terhelte le ezt a nyelvet, ami már nem tudta megfelelően kezelni őket. A 2.5.45-ös változat megjelenésekor ***Roman Zippel*** újraírta a beállítónyelvet, és új beállítóprogramok kerültek a fő rendszermagfába is. Az új beállítónyelv rugalmasságát és tudását tekintve egyaránt felülmúlja elődjét. A súgó szövegét is egyesíti a beállításokat vezérlő részekkel, így az illesztőprogramok foltjainak telepítése is könnyebb, és nem kell a központi ***Configuration.help*** fájllal való ütközések miatt aggódni.
A 2.5-ös sorozatban ***Kai Germaschewski*** és további kbuild-fejlesztők lassan átdolgozták a rendszermag `Makefile` rendszerét, így a rendszermagnak a kiválasztott beállításokkal való lefordítása is egyszerűbb lett. Írásomban a 2.5-ös rendszermagban szereplő `Makefile` és beállítófájlok formátumát, illetve az illesztőprogramok hozzáadásának és a rendszermag lefordításának a módját ismertetem.

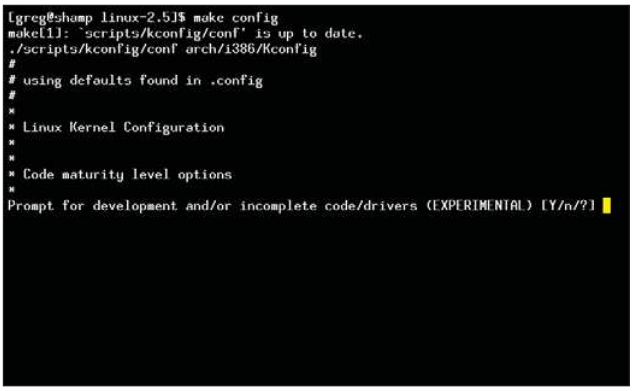
A rendszermag beállításai

A rendszermag beállításait szöveges vagy grafikus beállítóprogrammal adhatjuk meg. A szöveges beállítóprogram a `make config` paranccsal indítható; a beállítási lehetőségeket sorban egymás után kínálja fel a felhasználónak (*1. kép*). Az ugyancsak szöveges ncurses és változat jóval népszerűbb, indításához a `make menuconfig` parancsot kell kiadni (*2. kép*). A Qt alapú grafikus beállítóprogram a `make xconfig` paranccsal vehető használatba (*3. kép*).
A rendszermag beállításait kezelő program indításkor kiolvassa a fő beállítófájl – i386 alapú gépeknél az ***arch/i386/Kconfig*** – tartalmát. A többi géptípus is saját beállítófájlokkal bír a megfelelő könyvtárakban. A fő beállítófájl további állományokat is magába olvaszt a rendszermag könyvtárfájában lévő különféle alkönyvtárakból. Ezek a fájlok hasonló módon további állományokat foglalhatnak magukba. Az ***arch/i386/Kconfig*** fájlban például az alábbi sor szerepel:

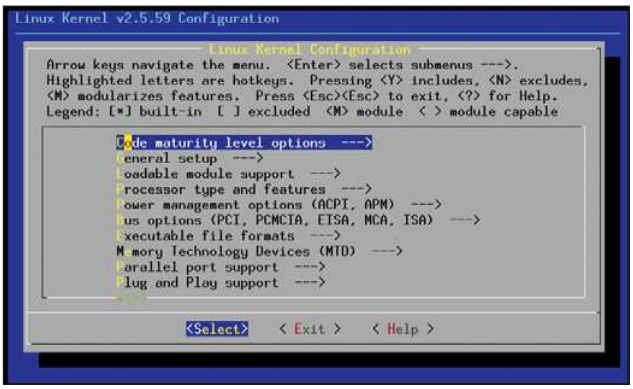
```
source "sound/Kconfig"
```

Vagyis beillesztjük a fenti állomány tartalmát. A ***sound/Kconfig*** fájl ugyanakkor több más állományt is magában foglal:

***source** "sound/core/Kconfig"*
***source** "sound/drivers/Kconfig"*



1. kép A rendszermag beállításainak megadása a make config segítségével

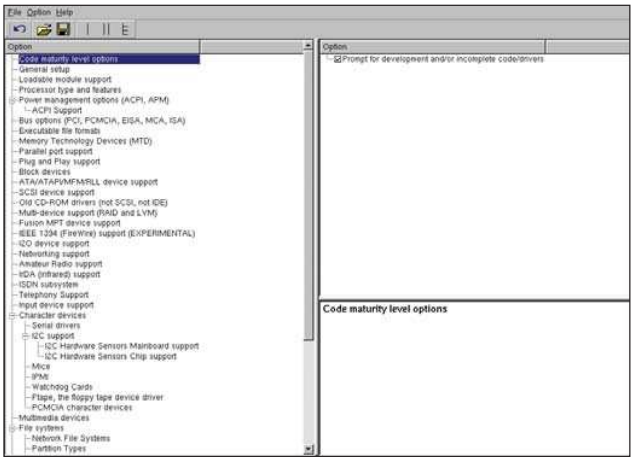


2. kép A make menuconfig használatakor könnyebb visszalépni és kijavítani a hibákat

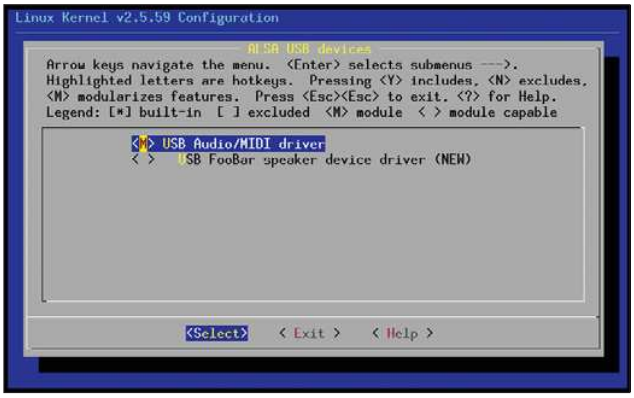
***source** "sound/isa/Kconfig"*
***source** "sound/pci/Kconfig"*
***source** "sound/ppc/Kconfig"*
***source** "sound/arm/Kconfig"*
***source** "sound/usb/Kconfig"*

A ***sound/usb/Kconfig*** fájl az összes olyan beállítást tartalmazza, ami az ALSA USB illesztőprogrammal kapcsolatos:

```
# ALSA USB drivers

menu "ALSA USB devices"
     depends on SND!=n && USB!=n

config SND_USB_AUDIO
     tristate "USB Audio/MIDI driver"
     depends on SND && USB
     help
       Say ·Y· or ·M· to include support for
       USB audio and USB MIDI devices
```



3. kép A Qt alapú make xconfig



4. kép Az újonnan hozzáadott FooBar USB hangszóró

endmenu

A Kconfig-fájlokban megjegyzéseinket a kettős kereszt karakter (#) után írhatjuk be. A mögötte, azonos sorban található szövegrészeket a beállítóprogram nem olvassa be.
A `menu` és `endmenu` parancsok a beállítóprogramot arra utasítják, hogy újabb menüszintet vagy képernyőt nyisson meg. A `menu` sorban a menü nevét idézőjelek közé szúrva kell megadni. A fenti fájlban a menü neve a következő: `"ALSA USB devices"`.
A menük és a beállítási lehetőségek esetében megadható, hogy megjelenjenek-e a képernyőn vagy sem. A fenti példában az USB-beállítások menüje csak akkor jelenik meg, ha a `CONFIG_SND` és a `CONFIG_USB` beállítást is kiválasztottuk; a függést a `depends on SND!=n && USB!=n` sor adja meg.
A szükséges gépelés mennyiségének csökkentése értékében minden beállítási lehetőség a `CONFIG` kulcsszóval kezdődik, amit a beállítónyelv nem használ. A beállításoknak a következő értékek adhatók meg:

- `y` – a beállítás engedélyezve van,
- `n` – a beállítás nincs engedélyezve,
- `m` – az adott dolog modulként kerül lefordításra.

Ha a `CONFIG_SND` és a `CONFIG_USB` beállítások nincsenek `n` értékre állítva (vagyis belefordítjuk őket a rendszermagba, vagy modulok lesznek), a `CONFIG_SND_USB_AUDIO` beállítás jelenik meg a felhasználó előtt. A beállítás a három lehetséges érték közül egyet vehet fel, vagyis „háromállású” kapcsolóként viselkedik. A felhasználó a következő szöveget látja: „USB Audio/MIDI driver:”

A háromállású USB Audio/MIDI driver beállítás

A beállítási lehetőségek megadásakor az alábbi típusok közül választhatunk:

- `bool` – a változó az `y` és az `n` (igen, illetve nem) értéket veheti fel.
- `tristate` – a változó `y`, `n` vagy `m` értéket kaphat.
- `int` – a változó bármilyen számértéket felvehet.

Ezt a beállítási lehetőséget egy `depends` logikai sor szabályozza, ami a menüpontokéval azonos logika szerint működik. A `CONFIG_SND_USB_AUDIO` lehetőség a `CONFIG_SND` és a `CONFIG_USB` beállításoktól függ, vagyis ha ezek valamelyikét modulnak állítjuk be, akkor a `CONFIG_SND_USB_AUDIO` beállítás is modulértéket vesz fel. Ha mindkét vezérlőbeállítás le van tiltva (vagyis `n` értéket kaptak), a függő beállítási lehetőség nem jelenik meg. Ha mindkettőnek `y` értéket adunk, a függő beállítás `n`, `y` vagy `m` értéket kaphat. Ezt a viselkedést egyetlen sor írja elő:

```
depends on SND && USB
```

A rendszermag kódjából a beállítási érték látható lesz (jelen esetben a `CONFIG_SND_USB_AUDIO`), így a kód ennek és bármely beállítási értéknek a meglétét ellenőrizni tudja. Meg kell jegyezni, hogy a `#ifdef`-nek az egy *.c* fájlon belül valamilyen beállítási érték ellenőrzésére történő használata ellenkezik a rendszermagra vonatkozó programozási stílusbeli alapelvekkel. Ezzel már korábban is foglalkoztam egy cikkemben, lásd „Proper Linux Kernel Coding Style” (Linux Journal, 2002. július, ➲ http://www.linuxjournal.com/article/5780).
Az `#ifdef` parancsokat csak a *.h* fájlokban használjuk, a *.c* fájlok maradjanak tiszták és könnyen olvashatók.
A súgó szövege korábban egyetlen nagy ***Configuration.help*** fájlba került. Most a leírás a Kconfig állományban, közvetlenül a `depends` sor után található. Egy `help` vagy `---help---` tartalmú sorral kezdődik, ezt további sorok követik, a `help` sorhoz képest két szóközzel beljebb.

Új beállítási lehetőség hozzáadása

Ha új beállítási lehetőséget kívánunk hozzáadni, a meglévő Kconfig fájlt csupán a kívánt helyen a megfelelő sorokkal kell bővítenünk. Ha például új USB illesztőprogram készül az ALSA hangrendszerhez, akkor az a ***sound/usb*** könyvtárba fog kerülni, és egy ***sound/usb/Kconfig*** fájlt kell hozzáadni. Az alábbi illesztőprogram a képzeletbeli FooBar USB-s hangszóró vezérlését végzi. A `CONFIG_SND` és a `CONFIG_USB` beállítások engedélyezésétől, továbbá a `CONFIG_SND_USB_AUDIO` beállítástól függ, ugyanis az új illesztőprogram felhasználja ennek az illesztőprogramnak bizonyos szolgáltatásait. Az új beállítási lehetőség az `SND_USB_AUDIO` pont után kap helyet, de még a záró `endmenu` parancs előtt, valahogy így:

```
config SND_USB_FOOBAR
    tristate "USB FooBar hangsz r
illesztıprogramja"
    depends SND_USB_AUDIO
    help
        `ll tsd Y ØrtØkre, ha FooBar USB
```

```
        hangsz r t szeretnØl hasznÆlni.

        A k d modulkØnt is hasznÆlhat .
        (= Ilyenkor sz ksØg szerint
illeszthetı
        a fut  rendszermaghoz, illetve
eltÆvol that .)
        A modul a k vetkezı nevet kapja:
        usbfoobar.o
```

A beállítási lehetőség akkor jelenik meg, ha az `SND_USB_AUDIO` beállítást választottuk ki (*4. kép*).

A rendszermag fordítása

A rendszermag fordítása különálló `Makefile` állományok alapján történik, ezek összekapcsolása fordítás közben történik meg, így egy nagyméretű `Makefile` jön létre.
A különálló `Makefile` állományok a megszokottól eltérő, kifejezetten a rendszermag fordítási folyamatára jellemző formátumot követnek. A `Makefile` feladatköre a szükséges fájlok – a megadott beállításoktól függő, megfelelő formátumú, vagyis az eredmény vagy modul lesz, vagy beépül a rendszermagba – létrehozatalára korlátozódik. Például a 2.5.59-es rendszermagbeli ***drivers/usb/misc/Makefile*** állomány így néz ki:

```
#
# Makefile for the rest of the USB drivers
# (the ones that don·t fit into any other
# categories)
#

obj-$(CONFIG_USB_AUERSWALD)    += auerswald.o
obj-$(CONFIG_USB_BRLVGER)      += brlvger.o
obj-$(CONFIG_USB_EMI26)        += emi26.o
obj-$(CONFIG_USB_LCD)          += usblcd.o
obj-$(CONFIG_USB_RIO500)       += rio500.o
obj-$(CONFIG_USB_SPEEDTOUCH)   += speedtch.o
obj-$(CONFIG_USB_TEST)         += usbtest.o
obj-$(CONFIG_USB_TIGL)         += tiglusb.o
obj-$(CONFIG_USB_USS720)       += uss720.o

speedtch-objs := speedtouch.o atmsar.o
```

Amennyiben a `CONFIG_USB_LCD` beállítás `m` értéket kapott, az ***usblcd.c*** modulba történő fordításáról az alábbi sor gondoskodik:

```
obj-$(CONFIG_USB_LCD)          += usblcd.o
```

Egyéb esetben, ha a beállítás `y` értéket kapott, a kódrészlet közvetlenül a rendszermagba épül. Ha a megfelelő modul egyetlen ***.c*** fájlból jön létre, akkor mindössze ennyi kell ahhoz, hogy új rendszermag `Makefile`-t hozzunk létre.
Ha az illesztőprogram több ***.c*** állományból épül össze, az állományok neveit különálló sorokban kell felsorolni, az illesztőprogram, avagy a modul nevével párosítva. Az előbbi példában a fájlok és az illesztőprogramnevek listája a következőképpen alakul:

```
obj-$(CONFIG_USB_SPEEDTOUCH) += speedtch.o
```

és a másik a

```
speedtch-objs := speedtouch.o atmsar.o
```

Az első sor adja meg, hogy a `speedtch` modul létrejöjjön-e. Ha igen, a sor jelzi, hogy a rendszermagba kerül-e vagy modul lesz. A második sor tudatja, hogy a ***speedtouch.c*** és az ***atmsar.c*** fájlok ***.o*** fájlokba fordítandók, és összekapcsolva fogják alkotni a `speedtch.o` modult.
A régebbi rendszermagoknál, ha egy fájl szimbólumokat is közzétett, akkor közvetlenül is szerepelnie kellett a rendszermag `Makefile` állományaiban. A 2.5-ös és újabb rendszermagoknál ez a kötelezettség már nem áll fenn.

Új illesztőprogram hozzáadása a fordítási folyamathoz

Ha új illesztőprogrammal szeretnénk bővíteni a rendszermag fordításának folyamatát, és az illesztőprogram egyetlen fájlban található, akkor mindössze egy sorra van szükségünk.
A FooBar USB hangszóró példájára utalva az

```
obj-$(CONFIG_SND_USB_FOOBAR) += usbfoobar.o
```

sort kell hozzáadni a ***sound/usb/Makefile*** állományhoz.
Ha az illesztőprogram két fájlban található, például a ***foobar1.c*** és a ***foobar2.c*** fájlban, akkor egy további sorra is szükség van:

```
usbfoobar-objs := foobar1.o foobar2.o
```

Összegzés

A 2.5-ös rendszermagnál a beállítások megadásának, illetve a fordításnak a folyamata jelentősen leegyszerűsödött és rugalmasabbá vált a korábbi változatokhoz képest. Roman Zippel és Kai Germaschewski munkájának hála a rendszermag fejlesztői több figyelmet fordíthatnak magára a programozásra, és kevesebbet kell törődniük a rendszermag fordításának bonyolult folyamatával.
A Kbuild folyamatról további tudnivalókat ***Sam Ravnborg*** kiváló írásában találhatunk, amely a
➲ http://marc.theaimsgroup.com/?l=linux-kernel&m=104162417329638 címen érhető el.

Linux Journal 2003. május, 109. szám

Greg Kroah-Hartman (greg@kroah.com)
Jelenleg a Linux USB és a PCI Hot Plug rendszermag felelőse. Az IBM-nél dolgozik, ahol számos, a Linux rendszermagjával kapcsolatos kérdéssel foglalkozik.

KAPCSOLÓDÓ CÍMEK

A Linux-rendszermag otthona
➲ http://www.kernel.org

A 2.5-ös fejlesztői rendszermag letöltési helye
➲ ftp://ftp.kernel.org/pub/linux/kernel/v2.5/

Rendszermag-levelezési lista
➲ http://www.cs.helsinki.fi/linux/linux-kernel/

2.5-ös rendszermag-beállítási tippek
➲ http://www.linux-sxs.org/administration/25xconf.html

A magmódú Linux

Modulkészítés nélküli programfuttatás a magtérben.

A Kernel Mode Linux (KML) olyan módszer, ami az egyszerű felhasználói térben futó programok számára lehetővé teszi, hogy a magtérben fussanak. Írásunk ennek a hátterét világítja meg.

A hagyományos rendszermagok a processzor beépített szolgáltatásai révén védik magukat. A Linux-rendszermag például a saját védelmére a processzor legkiváltságosabb szintjét és memóriavédelmi szolgáltatásait használja. A rendszermag saját magának a legkiváltságosabb szintet tartja fenn: a magmódot. Ezzel szemben a felhasználói folyamatok a legkevésbé kiváltságos szinten, felhasználói üzemmódban futnak. Ilyeténképpen maga a processzor védi a rendszermagot, minthogy a felhasználói módban végrehajtott programok nem férhetnek hozzá olyan tárterületekhez, amelyek a magmódban végrehajtott programokhoz tartoznak.

Mivel a felhasználói folyamatok nem képesek rendszerhívásokat kiadni a rendszermagban, ezért e nehézség feloldásához a hagyományos rendszermagok a korszerű processzorok nyújtotta szolgáltatásokat aknázzák ki, biztonságosan, de korlátozott mértékben növelve a rendszer kiváltságos üzemmódjának a szintjét. Vegyük példaként az IA-32 felület számára készült Linux-rendszermag programvezérelt megszakításkezelő (software interrupt) eljárását.

A programvezérelt megszakítás különleges ugróutasításnak tekintendő, aminek a célcímét a rendszermag korlátozza.

A kezdeti értékadás során a programvezérelt megszakítás célcímét a rendszerhívásokat kezelő különleges eljárás címére állítja be. Rendszerhívások igénybevételéhez a felhasználói program egy különleges utasítást, az `int 0x80`-at hajtja végre, ekkor a rendszermagban lévő rendszerhívásokat kezelő eljárás magmódban lesz végrehajtva. Az eljárás környezetet vált, azaz menti a felhasználói program regisztereinek tartalmát. Végül meghívja azt a magszolgáltatást, ami a felhasználói program által kért rendszerszolgáltatást kivitelezi.

Egy mostani Pentium 4-es gépen a programvezérelt megszakításkezelés és környezetváltás mintegy 132-szer lassabb, mint egy puszta függvényhívás.

Mellesleg az IA-32 számára készült, 2.5.53 és nagyobb változatszámú Linux-rendszermagok néhány különleges utasítást használnak a rendszerhívásokhoz: a `sysenter`-t és a `sysexit`-et. Ez azonban még mindig 36-szor lassabb, mint egy egyszerű függvényhívás.

A rendszerhívások felgyorsításának kézenfekvő módja a felhasználói folyamatok magmódban történő végrehajtása. Ekkor a rendszerhívások kezelése felgyorsul, hiszen a programvezérelt megszakítások és környezetváltások szükségtelenek. Ezek ettől kezdve már nem többek egyszerű függvényhívásoknál, mert a felhasználói program most már közvetlenül hozzáfér a rendszermaghoz. Úgy tűnik, hogy ez a megközelítés biztonsági réshez vezet: a magmódban végrehajtott felhasználói programok a rendszermag tetszőleges részéhez hozzáférhetnek.

A statikus programelemzés területén elért eredmények, például a típuselmélet (type theory) felhasználhatók a rendszermag felhasználói programokkal szembeni védelmére. Számos módszer támogatja ezt a programalapú védelmi megközelítést, többek között a Java bytecode, a .NET CIL, a O'Caml, a Typed Assembly Language és a Proof-Carrying Code.

KML: felhasználói folyamatok végrehajtása magmódban

A rendszermag által védett program megalkotásához tett első lépésként kidolgoztam a KML-t. A KML tulajdonképpen a felhasználói folyamatokat magmódban futtató módosított mag, amit azután a magmódú felhasználói folyamatok hívnak meg. A magmódú felhasználói folyamatok közvetlen kölcsönhatásban állnak a rendszermaggal. Emiatt a rendszerhívásokkal kapcsolatos többletráfordítás elhanyagolható.

A KML az eredeti Linux-rendszermaghoz foltként áll rendelkezésre, emiatt a rendszermagot újra össze kell építeni. A KML használatához szükséges a folt, valamint a rendszermag beállításakor be kell kapcsolni a Kernel Mode Linuxot (KML). Építsük újra a rendszermagot, majd telepítsük, ezt követően indítsuk újra a gépet. A KML-folt a ➲ http://www.yl.is.s.u-tokyo.ac.jp/~tosh/kml címről tölthető le, illetve megtalálható a 48. CD Magazin/Magmod könyvtárában.

A legfrissebb KML-ben a */**trusted*** könyvtárban található programok magmódú felhasználói folyamatokként futnak; maga a rendszermag nem végez semmiféle biztonsági ellenőrzést. Vegyük például az alábbi parancsot:

```
% cp /bin/bash /trusted/bin &&
➥/trusted/bin/bash
```

Ez a bash parancsértelmezőt magmódban futtatja.

Mire képesek a magmódú felhasználói folyamatok?

A magmódú felhasználói folyamatok egyszerű felhasználói folyamatok, kivéve természetesen a kiváltsági szintjüket (privilege level). Éppen ezért ezek mindent meg tudnak tenni, amire az egyszerű felhasználói folyamatok képesek, például valamennyi rendszerhívásfajtát igénybe tudják venni, még a `fork`-ot (elágazás), a `clone`-t és az `mmap`-et is. Ezenkívül ha a legfrissebb GNU C könyvtárat – ezen a 2.3.2-es vagy ennél frissebb változatot vagy a CVS-ből származó fejlesztői változatot kell érteni – a magmódú felhasználói programokban a rendszerhívások önműködően függvényhívásokká alakítják, akkor akad néhány kivétel is, ilyen a `clone`. Ezért a felhasználói programodban levő rendszerhívásokból eredő többletráfordítás úgy lett felszámolva, hogy magát a programot nem is kellett hozzá módosítani.

A lapozási eljárás az egyszerű felhasználói folyamatokkal megegyező módon működik, vagyis a magmódú felhasználói folyamatok mindegyike saját címterülettel rendelkezik. Azon kívül, ha a magmódú felhasználói folyamatok feleslegesen sok memóriát foglalnának is le, a rendszermag önműködően kilapozza a tárból, ahogyan ezt az egyszerű felhasználói folyamatok esetében is tenné.

A kivételek, tehát a szakaszolási hibák (segmentation faults), érvénytelen műveleti kivételek (illegal operation exceptions) a felhasználói folyamatokkal megegyező módon kezelhetők, hacsak a program nem jogosulatlanul szeretne hozzáférni a

1. táblázat **Kísérleti környezet**

Processzor	Pentium 4 2,533 GHz (L2 gyorstár 512 KB)
Tár	1 GB (PC 2100 DDR SDRAM)
Merevlemez	80 GB
Operációs rendszer	Linux-rendszermag 2.5.59 (KML_2.5.59_001)
Libc	alfaváltozat glibc-2.3.2 a CVS-fából

2. táblázat **A rendszerhívások várakozási ideje (processzorciklusokban kifejezve)**

	Eredeti Linux	Magmódú Linux (KML)
getpid	432	12
gettimeofday	820	404

3. táblázat
Az újraolvasás átviteli sebessége: az átmeneti tár mérete = 8 KB

Áll. méret (KB)	Eredeti Linux (sysenterrel)	Magmódú Linux (KML)
16	2675	3223
32	2918	3188
64	3056	3365
128	2906	3205
256	2327	2486

(iozone -S 512 k -s 16 k -s 32 k -s 64 k -s 128 k -s 256 k -r 8 k) (Egység: MB/sec)

magból a tárterülethez, vagy hibásan hajt végre kiváltságokhoz kötött utasításokat (privileged instructions).
A példa kedvéért végezzük el a következő program összeépítését, és hajtsuk végre magmódú folyamatként:

```
int main(int argc, char* argv[])
{
    *(int*)0 = 1;

    return 0;
}
```

A folyamat végrehajtása szakaszolási hibával zárul, ez azonban nem jár együtt magpánikkal (kernel panic). Ez a példa egyúttal azt is megmutatja, hogy a jelző eljárás működik. A második példánkban építsük össze az alábbi programot és hajtsuk végre magmódú felhasználói folyamatként:

```
int main(int argc, char* argv[])
{
    for (;;);

    return 0;
}
```

Használjuk a CTRL-C billentyűket a folyamatnak való `SIGINT` küldéséhez. Figyeljük meg, hogy a folyamat meg is kapja a jelet, és rendesen befejeződik a végrehajtása. A második példánk azt szemlélteti, hogy a folyamatütemezés működik – vagyis még akkor is, ha egy magmódú felhasználói folyamat végtelen ciklusba kerül, az előjogait érvényesíti a folyamaton (leállítja), és más folyamatokat hajt végre. Talán már észre is vetted, hogy a rendszer továbbra is rendesen működik, még a példában levő végtelen ciklus ellenére is.

Milyen feladatok elvégzésére képtelenek a magmódú felhasználói folyamatok?

Annak ellenére, hogy a magmódú felhasználói folyamatok egyszerű felhasználói folyamatok, akad néhány őket érintő korlátozás is. Ha a magmódú felhasználói folyamatok megsértik ezeket a korlátozásokat, a rendszer meghatározatlan állapotba kerül. Ez a legrosszabb forgatókönyv szerint rendszerösszeomlást jelent. Lássuk az első korlátot!
Ne módosítsd a CS, DS, SS és FS szakaszregiszterek tartalmát. Az IA-32 számára készült KML feltételezi, hogy ezeket a regisztereket a magmódú felhasználói folyamatok nem módosítják; belül használja őket. A második korlát pedig az, hogy kiváltsághoz kötött műveleteket (privileged actions) ne hajts végre helytelen módon. Magmódban a programoknak bármilyen kivételezett műveletet jogukban áll végrehajtani. Abban az esetben azonban, ha ez a művelet az adott rendszermaggal nincs engedélyezve, a rendszer meghatározatlan állapotba kerül. Erre az esetre mutat példát a következő magmódú felhasználói folyamat:

```
int main(int argc, char* argv[])
{
    /* disable hardware interrupts */
    __asm__ __volatile__ ("cli");

    for (;;);

    return 0;
}
```

Ekkor a rendszer működése megakad. Tapasztalatom szerint csak kevés alkalmazás sérti meg ezeket a korlátokat. A korlátok megsértői között találjuk a WINE-t és a VMware-t is. Ezek a korlátok csak a magmódú felhasználói folyamatok számára állnak fenn. Az egyszerű felhasználói folyamatokat sohasem sújtják ezek a korlátok, még akkor sem, ha KML-képes rendszermagot futtatunk.

KML-belügyek

Az IA-32 processzorokban a futó program kiváltsági szintjét annak a kódrészletnek a kiváltsági szintje határozza meg, ahol a program végrehajtódik. Emlékezzünk csak vissza, hogy az IA-32 processzorok egyrészt egy szakaszból állnak, amelyet a CS elnevezésű szakaszregiszter jelöl ki, másrészt a szakaszba mutató eltolásból (offset), amit az EIP regiszter határoz meg. A kódszakasz kiváltsági szintjét tehát a szakaszleíró határozza meg. A szakaszleírónak van egy mezője, ami a szakasz kiváltsági szintjét határozza meg.
A Linux-rendszermag alapvetően két szakaszt hoz létre: a rendszermag kódszakaszát és a felhasználói kódszakaszt. A rendszermagkódszakaszt maga a rendszermag használja, kiváltsági szintje pedig magmódra (kernel mode) van állítva. A felhasználói kódszakasz egyszerű felhasználói folyamatoknak van fenntartva, kiváltsági szintje pedig felhasználói módra van beállítva (user mode). Amikor az `execve`-t felhasználói folyamat futtatására használjuk, az eredeti Linux-rendszermag a szakaszregisztert a felhasználói kódszakaszra állítja be. Így aztán a felhasználói folyamat valódi felhasználói üzemmódban

lesz végrehajtva. Ahhoz viszont, hogy a felhasználói folyamatot magmódú felhasználói folyamatként hajthassuk végre, a KML a folyamathoz tartozó CS regisztert – a felhasználói kódszakasz helyett – rendszermag kódszakaszra állítja. Ezután a folyamat magmódban hajtódik végre. A KML egyszerű megközelítése miatt a magmódú felhasználói folyamat átlagos felhasználói folyamat lehet.

A vereméséghiba és megoldása

A KML alapvető megközelítése egyszerű, a benne rejlő legnagyobb hibát vereméségnek nevezik. Mindenekelőtt azt fejtem ki, hogy az eredeti Linux-rendszermag miként kezeli a kivételeket – a lapozási hibákat és a megszakításokat időzítő megszakításokat – az IA 32 processzorokon. Ezt követően ismertetem a vereméséget, végül pedig az általam kidolgozott megoldást mutatom be. Az eredeti Linux-rendszermagban a megszakításkezelést a megszakításleíró táblában (Interrupt Descriptor Table, azaz IDT) kapuként meghatározott megszakításkezelő eljárások végzik. Ha megszakítási kérelem érkezik, az IA-32 felfüggeszti a program futását, menti a program végrehajtási környezetét, és végrehajtja a megszakításkezelő eljárást. A program kiváltsági szintjétől függ, hogy az IA-32 a megszakítások érkezésekor milyen módon menti a futó program végrehajtási környezetét. Ha az IA-32 a programot felhasználói módban hajtja végre, a memória vermét önműködően átkapcsolja a magveremre, azután a végrehajtási környezetet menti a magverembe. Amennyiben a program végrehajtása magmódban történik, az IA-32 processzor nem kapcsol át a saját tárvermére és nem menti a környezetet (az EIP, EFLAGS, ESP, és az SS regiszter tartalmát) a futó program tárvermébe. De vajon mi történik akkor, ha a KML magmódú felhasználói folyamata hozzáfér a tárveremhez, amit a processzor laptáblái még ki sem osztottak? Először laphiba keletkezik, ekkor a processzor megpróbálja megszakítani a folyamatot, majd megkísérel elugrani az IDT-ben meghatározott hibakezelőhöz. Ezt a feladatot azonban képtelen végrehajtani, hiszen nincs verem, ahová a végrehajtási környezetet menteni tudná, ugyanis a folyamat végrehajtása magmódban történt, így a processzor sohasem képes a tárveremből a magverembe kapcsolni. A processzor e végzetes hiba jelzésére egy különleges védelmi hibát próbál meg létrehozni, az úgynevezett kettős hibát. Újra csak a régi hibával állunk szemben: a processzor nem tud kettős védelmi hibát létrehozni, mert nincs verem, ahová a futó folyamat végrehajtási környezetét menthetné. Végül a processzor feladja, és újraindítja a gépet.

A vereméség megoldásához a KML az IA-32 processzor feladatkezelő szolgáltatását használja. A szolgáltatás használatával a mag egyetlen utasítás segítségével képes váltogatni a folyamatok között. A jelenlegi rendszermagok azonban már nem használják ezt a szolgáltatást, mert a kizárólag programvezérelt megoldásoknál lassabb, így szinte feledésbe merült.

Az IA-32 processzorok esetében e szolgáltatás ereje abban rejlik, hogy a megszakítások és a kivételek kezelésére egyaránt használható. Az IA-32 processzor által kezelt feladatokat be lehet állítani az IDT-re. Amikor megszakítás keletkezik, egy feladat rendelődik hozzá, ez kezeli a megszakítást – a processzor a megszakított program végrehajtási környezetét a tárvermek helyett először a feladat adatszerkezetébe menti. Ezt követően a processzor az IDT-ben meghatározott feladat-adatszerkezetből helyreállítja a környezetet.

A legfontosabb az, hogyha a feladatkezelő szolgáltatást használjuk a megszakítások kezelésére, akkor a tárveremben nincs szükség kapcsolgatásokra, vagyis ha a laphibákat ezzel a programmal kezeljük, akkor a magmódú felhasználói folyamat képes biztonságosan elérni a tárvermet. Ha viszont az összes lapozási hibát ezzel a szolgáltatással kezeljük, akkor romlik a rendszer működőképessége, mivel a feladatalapú megszakításkezelés lassabb, mint az egyszerű megszakításkezelés. Ezért csak a kettős kivétel hibákat kezeljük ilyen módon, más szóval kizárólag a tárverem hiányból adódó hibák kezelését végzi ez a szolgáltatás. Tapasztalatom szerint a tárvermek csak ritkán okoznak lapozási hibákat, és a működéscsökkenés is elhanyagolható.

A működés mérése

A működés javulásának mérésére két kísérletet végeztem. Mindkét mérés az eredeti és KML rendszermag működését vette össze. Az eredeti Linux-rendszermag működésének mérésére a `sysenter/sysexit` módszert választottam a `0x80`-as utasítás helyett. A kísérleti környezet az *1. táblázatban* látható. Az első kísérletben a `getpid` és `gettimeofday` rendszerhívások várakozási idejét mértem, a rendszerhívásokat a felhasználói programok közvetlenül kezdeményezték a mérésekben, a libc nélkül. A várakozási időt az `rdtsc` utasítással mértem, az eredményeket pedig a *2. táblázatban* foglaltam össze.

A végeredmény azt tükrözi, hogy a `getpid` a KML-ben 36-szor volt gyorsabb, mint az eredeti Linux-rendszermagban; a `gettimeofday` pedig kétszer olyan gyors volt a KML-ben, mint az eredeti Linux-magban.

A második kísérlet állomány B/K-mérés volt, amit az Iozone állományrendszer-mérővel folytattam le. A be- és kivitelt négy fájlművelettípusra nézve vizsgáltam meg: írás, újraírás, olvasás és újraolvasás. A mérést különböző méretű állományokkal végeztem, aminek a mérete 16 KB-tól 256-KB-ig terjedt, az átmeneti tár méretét pedig 8 KB-ban rögzítettem.

A rendszer hátterét az `ext3` állományrendszer adta. Minden egyes Iozone-mérést harmincszor hajtottam végre, és a legjobb átviteli eredményt választottam.

Az újraolvasás átviteli teljesítménye a *3. táblázatban* található. Helyhiány miatt az írás, újraírás, olvasás és újraolvasás részletes eredményeinek bemutatásától eltekintünk.

Az eredmény azt mutatja, hogy KML-ben az újraolvasás sebessége 6,8–21 százalékkal javult. Ezenkívül az írás 0,6–3,2%-kal, az újraírás 3,3–5,3%-kal és az olvasás 3,1–15%-kal lett gyorsabb. Ezek a kísérleti eredmények azt jelzik, hogy a KML képes javítani a rendszerhívásokat gyakran igénybevevő alkalmazások működésén, amik kisebb állományokat olvasnak vagy írnak. Például a webkiszolgálók, adatbázisok hatékonyan üzemeltethetők KML-ben.

A megelőző kísérletekből érdemes megemlíteni, hogy KML kiiktatta a rendszerhívásokkal járó többletterhet. Az alkalmazáson végzett kevés finomítással a KML további teljesítményjavulást mutathat. Például a magmódú felhasználói folyamatok a további javulás érdekében képesek lesznek közvetlenül hozzáférni az átmeneti tárakhoz.

A kapcsolódó címek megtalálhatóak a 48. CD Magazin/Magmod könyvtárában.

Linux Journal 2003. május, 109. szám

Toshiyuki Maeda

A Tokiói Egyetemen számítógéptudományból készül PhD fokozatot szerezni. Kedvenc képregényei a Hikaru no GO (Hikaru go-játéka), Jojo no Kimio na Boken (Jojo bizarr kalandja), és Runatikku Zatsugidan (Bolondos akrobata-színtársulat).

A Linux-rendszermag titkosító API-ja

Új általános keretrendszerben érhetők el a titkosító szolgáltatások a rendszermag minden része számára.

Írásunkban a rendszermag új titkosító API-ját mutatjuk be röviden. Mindenkihez szólunk, akit izgat a Linux működése, például rendszergazda vagy olyan érdeklődő személy, aki szeretne bepillantani az API tervezésébe, megvalósításába és alkalmazásaiba.
A rendszermag működésének ismerete előnyt jelent, de ez nem szükséges ahhoz, hogy az API-t nagy vonalakban megértsük.
Ez az API csak rövid ideje létezik. Nem sokkal azután, hogy 2002 őszén a rendszermagot lezárták, vagyis az új szolgáltatásokat megvalósító kód beépítése nem volt lehetséges, a ***Dave Miller*** és ***Alekszej Kuznyecov*** által fejlesztett IPSec-megvalósítást jelölték a 2.6-os rendszermagba való beépítésre. Az IPSec a rendszermagon belül igényli a titkosítási szolgáltatásokat, és ez a rendszermag titkosító szolgáltatásaira való egyre növekvő általános igény mellett arra késztette a fejlesztőket, hogy új titkosító API-t fejlesszenek.

Tervezés

Bár eredetileg az IPSec támogatása volt a cél, az API-t általános célú szolgáltatásnak tervezték, amit a jövőbeli alkalmazások is kihasználhatnak, például az állománytitkosítók, a titkosított állományrendszerek, az állományrendszerek sértetlenségét felügyelő alkalmazások, a véletlenszám-előállító eszköz (***/dev/random***), a hálózati állományrendszerek biztonsága (például a CIFS) és egyéb hálózati szolgáltatások a rendszermagban, amelyek igénylik a titkosítást.
Az API tervezésekor kifejezetten szempont volt, hogy közvetlenül a lapvektorokon működjön. A lap a rendszermag által kezelt memória alapegysége. A lapvektorokon alapuló API lehetővé teszi a mély beépülést a rendszermag alszerkezeteibe, például a VFS-be vagy a hálózati verembe, valamint a szétszórás-összegyűjtés típusú műveletet is. Az IPSec esetében a titkosítási műveletek közvetlenül a hálózati csomagoknak megfeleltetett nem folytonos memórialapokra alkalmazhatók.
Az egyszerűség fontos tervezési szempont volt, ami általában is mindig jó, de különösen a rendszermag és a biztonsági szolgáltatások kódjánál lényeges.
A rugalmas használhatóság is a célkitűzések között szerepelt. Például az API rugalmasan kezeli az algoritmusokat: ezek magmodulként szükség szerint betölthetők, az API-nak semmit nem kell róluk előzetesen tudnia.
A jövőben a következőket szeretnék még megvalósítani:

- Titkosításgyorsító kártyák és az IPSec-es hálózati kártyák képességeinek a kihasználása.
- A felhasználó által előnyben részesített algoritmus kiválasztása, ha több lehetséges megvalósítás közül lehet választani (például hatékony gépi kódú megvalósítások vagy különböző alkatrészszintű megvalósítások).
- Az aszimmetrikus titkosítás (RSA) támogatása, amire a csoportszórásos IPSec és a magmodulok aláírása ellenőrzésének rendszermagbeli támogatásához lehet szükség. Ezen a területen sok vita várható, mert az aszimmetrikus titkosítás általában lassú és bonyolult – bármelyik a kettő közül elég okot ad arra, hogy kimaradjon a rendszermagból.
- Egységes API a felhasználói programok számára, amik az elérhető titkosítóeszközöket kívánják hasznosítani, például SSL, IPSec-kulcscsere, biztonságos útválasztó protokoll és DNSSEC.
- Az API memóriafoglalásának további csökkentése, hogy beágyazott eszközökben is használható legyen.

Algoritmusok

Jelenleg az API az alábbi háromféle algoritmust támogatja:

1. ***Kivonatolók*** (egyirányú hasítófüggvények) – ezek bemenete tetszőleges üzenet, kimenete rövid, adott méretű üzenetkivonat. Az egyirányúsághoz az szükséges, hogy a kimenet egyszerűen előállítható legyen, de az eredeti üzenetet ne lehessen könnyen visszaállítani belőle. A titkosítási célokra olyan hasítófüggvények felelnek meg, amik különböző bemenetekre ritkán adnak azonos kimenetet. A lehetséges alkalmazási területek között van az adatok sértetlenségének ellenőrzése és üzenethitelesítő kódok létrehozása hálózati protokollokhoz. Kivonatoló algoritmusokra példa az MD5 és az SHA1.
A HMAC nevű üzenethitelesítő séma (RFC2104) beépült az API-ba, és ez minden szabványos kivonatoló algoritmussal működik. Jelenleg az IPSec-csomagok hitelesítési adatainak előállítására használják.
2. ***Titkosítók*** – ezek az algoritmusok szimmetrikus kulcsú titkosítást valósítanak meg, ahol az üzenetet egy kulcs segítségével alakítják át titkosított üzenetté. Általában ugyanez a kulcs szolgál a titkosított üzenet visszafejtésére is. Követelménye, hogy az üzenetek titkosítása és visszafejtése a kulcs birtoklása esetén könnyű legyen (amit viszont titokban kell tartani), anélkül viszont nehéz. A lehetséges alkalmazási területek között szerepel az adattitkosítás és az üzenethitelesítő kódok létrehozása. Titkosító algoritmusokra példa a TripleDES, és a Blowfish, valamint az AES.
Kétféle titkosító létezik: a tömbtitkosítók adott hosszúságú adattömbökkel dolgoznak (például 16 bájt), a folyamtitkosítók kulcsfolyamot használnak, ami az adat egyetlen bitjén dolgozik egy lépésben. A titkosítók számos üzemmódban működhetnek, például ECB (Electronic Codebook) módban, amiben az üzenet minden tömbje egyszerűen van titkosítva a kulccsal, vagy CBC (Cipher Block Chaining) módban, amiben az előzőleg titkosított tömb a következő tömb titkosításában kerül felhasználásra.
3. ***Tömörítés*** – gyakran használják együtt a titkosítással, hogy nehezebb legyen kihasználni az eredeti üzenet gyengeségeit, és hogy gyorsabb legyen a titkosítás (ugyanis a tömörített üzenetek rövidebbek). A titkosított adatot lényegéből adódóan nem nagyon lehet tömöríteni, és ez hátrányosan befolyásolja a teljesítményt az olyan átviteli csatornák esetében, amelyek kihasználják a tömörítést. Ha az ada-

tokat még a titkosítás előtt tömörítik, sok esetben elkerülhető a teljesítménycsökkenés. Tömörítő algoritmusokra példa az LZS és a Deflate.
Eddig a jól ismert forrásokból származó algoritmusokat tették alkalmassá az API-val való használatra, mivel ezeket valószínűleg jobban átnézték és többen próbálták ki. A rendszermag főágába általában csak olyan algoritmus kerülhet be, ami nem áll szabadalmi oltalom alatt (azaz az IDEA 2011-ig nem kerülhet be), nyílt és elismert szabványokon alapul, és kipróbálására tesztcsomag áll rendelkezésre.

Lapvektorok

Mielőtt az API szerkezetének tárgyalásába kezdenénk, röviden tekintsük át a memórialapok és a lapvektorok témakörét. Mint az fentebb említettük, a lap a rendszermag által kezelt memória alapegysége (i386-on a lapok mérete 4 KB). Képzeljünk el egy tárolót, amiben mondjuk 1460 bájt felhasználói adat van. Ez a rendszermag adott lapjához tartozik, a lap kezdetétől számított bizonyos eltolási értéktől kezdődik, és 1460 bájt hosszú. Ezt a tárolót a következő lapalapú szerkezet írja le: `{ lap, eltolÆs, hossz }`.
A lapokkal közvetlenül dolgozó csatolónak, például a titkosító API-nak ezzel a szerkezettel, más néven a lapvektorral kell foglalkoznia. A megvalósítás egy már meglévő rendszermagbeli adatszerkezetet, az úgynevezett szórási listát használja. A szórási listában lapvektorok vannak, és általában szétszórásos–összegyűjtéses közvetlen memória-hozzáférési műveleteknél használatos.
A titkosító API arra használja a szórási listát, hogy nem folytonos lapvektorokon végezzen műveleteket. A rendszermagban a szétszórás–összegyűjtés elsődleges célja az adatok felesleges másolásának elkerülése. A tapasztalat szerint ettől a forráskód is áttekinthetőbb lesz. Számos olvasónak ismerős lehet a szétszórásos–összegyűjtéses I/O a `readv()` és a `writev()` rendszerhívások formájában. A rendszermag titkosító API-ja ugyanezt az általános elvet használja fel, de egyszerű memóriaterületek helyett lapokon működik.

Az API szerkezete

Az API két elsődleges objektummal foglalkozik:

- ***Algoritmusmegvalósítások*** – magmodulok, amik a szükséges algoritmusok kódját tartalmazzák.
- ***Átalakítók*** – objektumok, amik példányosítják az algoritmusokat, kezelik a belső állapotot és a közös megvalósítási logikát. Az átalakítókat a `crypto_alloc_tfm()` és a `crypto_free_tfm()` segítségével kezelhetjük. Az API-hoz tartozó burkolók segítségével leegyszerűsödik az átalakítók használata, és lehetővé válik az átalakító alatt meghúzódó algoritmus tulajdonságainak a lekérdezése.

A következő példakód az átalakító jellemző használatát mutatja be, ez a Blowfish-kódolóval ECB módban bizonyos rendszermagkód adatokat szeretne titkosítani:

```
tfm =
➥crypto_alloc_tfm("blowfish",
CRYPTO_TFM_MODE_ECB);
crypto_cipher_setkey(tfm,
➥key, keylength);
crypto_cipher_encrypt(tfm,
➥&scatterlist, numlists);
crypto_free_tfm(tfm);
```

Az ábrán (48. CD Magazin/Crypt könyvtár) látható, hogy az API réteges felépítésű, azaz a mag a titkosítás felhasználói és az algoritmusok megvalósítói elől rejtve van. Ez a mag tartalmazza az általános átalakítókezelést, a szórási lista kezelését és az algoritmusok elvonatkoztatását. Még lejjebb az algoritmustípus szerinti logika – például a titkosítók feldolgozási üzemmódja és a kivonatok használata az üzenethitelesítő kódok előállítására – található.
Az algoritmuskezelő réteg tartalmazza az algoritmusok megvalósításait felkutató, betöltő és a rájuk való hivatkozást számláló logikát. Az utóbbi azért szükséges, hogy elkerüljük csúnya dolgok bekövetkezését, amik akkor történnének, ha egy használatban levő algoritmusmodult törölnénk a memóriából. Létezik egy futásidejű algoritmuslekérdező felület is, amin keresztül a hívó kód meghatározhatja a rendszeren elérhető algoritmusokat. Ezt elsősorban a kulcstárgyalási protokollokhoz, például az ISAKMP/IKE-hez tervezték használni. Végül az algoritmusbejegyző felület a modulok számára lehetővé teszi, hogy egy vagy több algoritmust jegyezzenek be – különféle tulajdonságokat, mint például az algoritmus nevét, tömbméretét, a legkisebb és legnagyobb kulcshosszt megadva. Az adott pillanatban bejegyzett algoritmusok és tulajdonságaik a */proc/crypto* könyvtárban tekinthetők meg.

Adatok

A 2.5-ös rendszermagfában lásd:
- Documentation/crypto/
- include/linux/crypto.h
- crypto/

A ➲ http://samba.org/~jamesm/crypto címen található weboldal további tudnivalókat tartalmaz a fejlesztők számára, például a tennivalók listáját.

Összegzés

A titkosító API még fiatal, de lehet belőle valami, főleg ha a felsorolt jövőbeli tervezési célok egy részét megvalósítják.

Köszönetnyilvánítás

Köszönöm ***David Miller*** és ***Nancy Chan*** segítségét, akik átnézték ezt a cikket.

Linux Journal 2003. április, 108. szám

James Morris
Programfejlesztő – a Netfilter, az LSM, az SELinux és a Linux-rendszermag-titkosító API-projektekben dolgozik. Sydneyben független tanácsadó.

Halmozható fájlrendszerek készítése

Mostantól újraírás nélkül is új képességekkel bővíthetjük kedvenc fájlrendszerünket.

Fájlrendszert, de egyáltalán bármilyen rendszermagkódot írni meglehetősen nehéz feladat. A rendszermag összetett környezet, ahol egyetlen apró hiba is végzetes adatvesztéshez vezethet. A fájlrendszerek a felhasználói alkalmazások szemszögéből nézve átlátszó, egyszerű adatelérési felületet kínálnak, így a fejlesztők állandóan új képességekkel szeretnék bővíteni a fájlrendszereket. E cikkben gyors útmutatást adunk, ami alapján bárki anélkül adhat hozzá új képességeket meglévő fájlrendszeréhez, hogy rendszermag vagy fájlrendszertudor lenne.

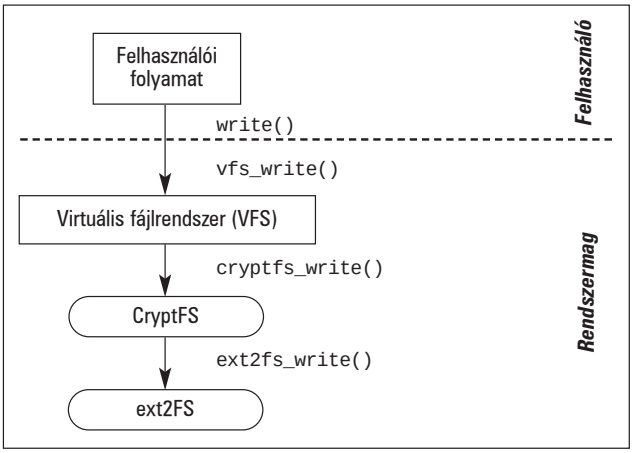
Szóval szeretnél fájlrendszerfejlesztő lenni?

A Linux több hasonló fájlrendszertípust is támogat: lemezes fájlrendszerek, hálózati fájlrendszerek stb. Egy hatékony és megbízható fájlrendszer kifejlesztése éveket vesz igénybe, és ha egyszer végre üzembiztosan fut, nem szívesen tennénk tönkre új szolgáltatások beiktatásával. Ezenkívül a fájlrendszerek karbantartói igen ritkán fogadnak el új képességeket vagy fejlesztési foltokat üzembiztos fájlrendszerükhöz. Így aztán egyáltalán nem meglepő, hogy a jelenleg használatos legnépszerűbb fájlrendszerek évek óta szinte semmit nem változtak. Tegyük fel, hogy egy egyszerű titkosító fájlrendszert szeretnénk készíteni, ami valamilyen előre megadott kulccsal titkosítja az adatot. A különféle kódolásokhoz könnyedén megszerezhetjük a hordozható C-kódot. Ezután a rendszerbe kell illesztenünk az adatvermet kódoló és dekódoló hívásokat. Elméletileg a feladat egyszerű: az írási rendszerhívásokból érkező összes adatot – mielőtt a lemezre írnánk – titkosítjuk, a lemezről érkező adatokat pedig visszafejtjük, még mielőtt a rendszerhívást kiadó folyamatnak visszaadnánk.

Ha módosításról van szó, az ember első gondolata az, hogy lemásolja az `ext2` 5000+ soros forráskódját, áttanulmányozza, majd beilleszti a kódolórészeket. De inkább álljunk ellen a késztetésnek, és ne másoljunk le egy egész fájlrendszert. Igaz ugyan, hogy mindössze 5000+ sorból áll, de a rendszermagkódot legalább egy nagyságrenddel bonyolultabb fejleszteni, mint a felhasználói szintű programokat. Mégha végül sikerül is kódoló eljárásainkat beilleszteni a megfelelő helyre, rá kell ébrednünk, hogy időnk nagy részét tanulmányozással töltöttük, s végül csak néhány helyen szúrtunk be pár sornyi kódot. És mit kaptunk eredményül? Egyetlen, titkosítással kibővített `ext2` fájlrendszert. Mi történik, ha a titkosítót inkább NFS fájlrendszerrel vagy valamely másik rendszer alatt szeretnénk használni a számtalan Linux-fájlrendszer közül?

Egymásra épülő fájlrendszerfejlesztés

A Linux, akárcsak a legtöbb operációs rendszer, két részre bontja a fájlrendszerkódot: vannak belső részei (`ext2`, NFS stb.), és van egy általános célú rétege, amit virtuális fájlrendszernek neveznek (VFS). A VFS réteg a rendszerhívások és a belső fájlrendszerek között helyezkedik el. A VFS célja egységes elérést biztosítani a különböző fájlrendszerekhez, elrejtve azok pontos részleteit. Amikor a fájlrendszerek betöltődnek a rendszermagba, beállítanak néhány függvénymutatót (OO-szóhasználat szerint metódust), amiket azután a VFS használhat. A VFS általában ezeket a függvényeket hívogatja, mit sem sejtve arról, hogy pontosan milyen fájlrendszerek függvényei rejlenek a mutatók mögött. Például az `unlink` rendszerhívás a `sys_unlink` szolgáltatáshívássá alakul, és meghívja a `vfs_unlink` VFS-függvényt, ez azután a feltelepített mutató alapján elindítja a fájlrendszerfüggő függvénymutatót: `ext2` esetében ez az `ext2_unlink` lesz, NFS esetében az `nfs_unlink`, más fájlrendszerek esetében pedig a számukra megfelelő függvényt. Ebben a cikkben a fájlrendszer függvénymutatókat a `->` jellel fogjuk jelölni, a következő formában: `->unlink()`.



1. ábra Példa a halmozható titkosító fájlrendszerre

Mivel mi a titkosító fájlrendszerünket gyorsan szeretnénk fejleszteni, a következő szállóigét alkalmazzuk: „A számítástechnikában egy újabb közvetítő szint felvételével minden feladat megoldható.” Szerencsére a Linux VFS lehetővé teszi, hogy a VFS és egy másik fájlrendszer közé újabb fájlrendszert szúrjunk be. Az *1. ábra* egy ilyen, CryptFS nevű halmozható (stackable) titkosító fájlrendszert mutat be. A CryptFS-t azért nevezzük halmozhatónak, mert egy másik fájlrendszer (`ext2`) tetejére pakoljuk rá. A VFS réteg a CryptFS `->write()` függvénymutatóját (`cryptfs_write`) hívja meg; a CryptFS titkosítja a felhasználói adatokat, fogadja, majd továbbadja az alatta elhelyezkedő `->write()` függvénymutatónak (`ext2_write`). A halmozható fájlrendszerek általában önmagukban is állhatnak, és bármilyen már létező fájlrendszert hordozó befűzési pont fölé felrakhatók; ez annyit jelent, hogy (halmozható) fájlrendszerünket csak egyszer kell kifejlesztenünk, és az valamennyi helyi (alacsonyszintű) fájlrendszeren (`ext2`, NFS stb.) működni fog. Továbbá a Linux 2.4.20-astól kezdve, a halmozható fájlrendszerek távoli NFS-ügyfelekre is biztonságosan átvihetők (az `nfs-utils-1.0` vagy újabb változat segítségével).

Hogyan működik a halmozható fájlrendszer?

A halmozható fájlrendszer alapszolgáltatása műveletet és értékeket átadni a lejjebb található fájlrendszernek. Az alábbi

kódkivonatban bemutatjuk, hogyan kezeli a WrapFS nevű, egyszerű, valóságos feladatnélküli adattovábbító fájlrendszer az `->unlink()` műveletet:

```
int wrapfs_unlink(struct inode *dir,
                  struct dentry *dentry)
{
  int err = 0;
  struct inode *lower_dir;
  struct dentry *lower_dentry;

  lower_dir = get_lower_inode(dir);
  lower_dentry = get_lower_dentry(dentry);

  /* pre-call code can go here */
  err = lower_dir->i_op->unlink(lower_dir,
                        ➥lower_dentry);
  /* post-call code can go here */

  return err;
}
```

Amikor a VFS szeretne letörölni egy fájlt a WrapFS fájlrendszerről, meghívja a `wrapfs_unlink` függvényt, és átadja a törlendő fájlt tartalmazó könyvtár fájlleíró (inode) adatát (`dir`), valamint a törlendő bejegyzés nevét (a dentry belsejében található).
Minden egyes fájlrendszer nyilvántart néhány szükséges objektumot, például fájlleíró-azonosítókat, könyvtárbejegyzéseket, könyvtárneveket és nyitott fájlokat. Amikor halmozást alkalmazunk, akár több objektum is jelölheti ugyanazt az állományt – csak éppen különböző rétegeken. Például a ábrán látható CryptFS lehet, hogy fenntart egy könyvtárbejegyzés-(dentry) objektumot a fájlnév nyílt szöveges változatával feltöltve, míg az `ext2` ugyanennek a névnek a kódolt változatát tárolja egy másik könyvtárbejegyzésben. Amennyiben a halmozható fájlrendszer igazán átlátszó akar maradni a VFS és a többi fájlrendszer számára, kénytelen minden szinten több objektumot is fenntartani.
Ez az oka annak, hogy a `wrapfs_unlink` első dolga a megkapott értékek alapján megkeresni az adott objektumhoz tartozó könyvtárbejegyzés fájlleíróját az egy szinttel lejjebb befűzött fájlrendszeren. Ezek a `get_lower_*` függvények lényegében a WrapFS objektumok (létrehozásakor kitöltött) saját mezőiben található, előre tárolt mutatókat követik. Az alsóbb objektumok beazonosítását követően megkezdődhet a halmozás lényegi része. Meghívjuk az alacsonyabb szinten található fájlrendszer `->unlink()` függvénymutatóját az alacsonyabb szinten elhelyezkedő könyvtár fájlleírója alapján, és átadjuk a két alacsonyabb szintű objektumot.
A WrapFS teljes értékű nullréteg (vagy loopback) fájlrendszer, ami egyszerűen (és minden módosítás nélkül) csak műveleteket és objektumokat közvetít a VFS és az alul elhelyezkedő fájlrendszer között. Sajnos azért mégsem olyan könnyű dolog megírni a WrapFS-t, ugyanis az alsó fájlrendszerrel szemben VFS-ként kell viselkednie, ugyanakkor a Linux igazi VFS-rétege számára alsóbb fájlrendszernek kell látszania. Ez a kettős szerep a kulcsok, hivatkozásszámlálók és a memóriafoglalás óvatos használatát követeli meg. Szerencsére valaki volt olyan jó, és előre elkészítette, valamint fenntartja nekünk a WrapFS-réteget. A WrapFS-réteget nagyszerűen fel tudjuk használni mintaként, ha módosításokat vagy új szolgáltatásokat szeretnénk megvalósítani.

Nekirugaszkodás

Most már világos, hogyan működik a halmozás, de mi a következő lépés? Először is keressük meg azokat a helyeket a WrapFS-ben, ahová a saját kódunkat beilleszthetjük. Az imént bemutatott `wrapfs_unlink` kódhoz visszatérve három ilyen helyet találunk, az alsóbb szintű `->unlink()` függvénymutató előtt, után vagy helyett.

1. Előhívás: kódunkat beilleszthetjük az alacsonyabb szintű `->unlink()` hívás elé. Például ellenőrizhetjük, hogy a felhasználó fontos fájlt akar-e törölni, és ha igen, megakadályozhatjuk:
```
if (strcmp(dentry->d_name.name,
           "vmlinuz") == 0)
return -EACCES;
```
2. Hívás: az egész hívást is helyettesíthetjük. Például a fájl törlése helyett esetleg csak át akarjuk nevezni valamilyen egyszerű kis visszafordítható (undo) fájlrendszer részeként (valószínűleg mindannyian találkoztunk már a véletlenül kiadott `rm -f` parancsok hatásával).
3. Hívás után: itt az alsóbb fájlrendszerből visszatérő főművelet után tudunk egyéb műveleteket végrehajtani. Tegyük fel, hogy egy rossz szándékú felhasználó törölni akarja a */etc/passwd* fájlt, de a Unix jogosultsági rendszere ezt megakadályozza. A rendszergazda naplózni szeretné (a `syslogd` segítségével) az ilyesfajta eseményeket:
```
if (err == -EACCES &&
    strcmp(dentry->d_name.name,
           "passwd") == 0)
  printk("uid %d tried to delete passwd",
      current->fsuid);
```

A `current` mindig az éppen futó folyamatra mutató általános hatókörű (global) változó, az `->fsuid` pedig e folyamat azonosítója, amit a fájlrendszerek felhasználhatnak.
A fenti és az ezután következő példákat némileg leegyszerűsítettük, hogy a lényegi részek jobban láthatók legyenek, és kevesebb helyet foglaljanak. Például a `d_name.name` összetevő nem null végződésű (null terminated), így megfelelő hosszúságú `memcmp` utasítást kellene használni; illetve annak ellenőrzésére, hogy a dentry által megjelölt fájl valóban a */etc/passwd* állomány, meg kellene néznünk, hogy valóban ez a fájlrendszer-e a gyökérrendszer, vagy a `d_path()` segítségével meg kellene vizsgálnunk az abszolút elérési utat.
A 2.4.20 alatt kipróbált teljes példákat a FiST-honlapon, illetve a 48-as CD Magazin/FiST könyvtárában találjuk meg (➲ http://www.cs.sunysb.edu/~ezk/research/fist).

Ki felügyeli a kukkolókat?

A Unix megpróbálja védelmezni a fájlokat a jogosulatlan felhasználóktól. Amikor a felhasználó olyan fájlt akar megnyitni, amire nincs jogosultsága, a Unix azon nyomban egy „jogosultság megtagadva” hibaüzenetet ad vissza. Néhány felhasználó szeret mások állományai között nézelődni, néha kifejezetten olyan fájlokat keresve, amik véletlenül maradtak védtelenek, vagy megpróbálja kitalálni az esetleg létező fájlneveket valamelyik nem kereshető könyvtárban. Sajnos, mégha ezek a kukkolók nem is járnak sikerrel, az áldozatok többnyire nem sejtik, hogy ilyen jellegű próbálkozás történt.
Az egyik leggyakrabban használt fájlrendszerművelet a `->lookup()`, ami a rendszer minden egyes fájlnévhasználata során meghívódik. A rendszermagnak át kell alakítani a nevet (a karaktersorozatot) a tényleges VFS-objektum azonosítóne-

vére, vagyis inode, dentry vagy fájl alakba. Hogy kiszűrhessük a kukkoló felhasználókat, a következő kódot illesztettük a `snoopfs_lookup` vagy `snoopfs_permission` eljárásokba, közvetlenül az alsó fájlrendszer `->lookup()` hívása után:

```
if ((err == -EACCES ||
     err == -ENOENT) &&
    dir->i_uid != current->fsuid &&
    current->fsuid != 0)
  printk("snoop uid=%d pid=%d file=%s",
         current->fsuid, current->pid,
         dentry->d_name.name);
```

Megvizsgáljuk az alsóbb fájlrendszer `->lookup()` függvényétől visszakapott hibakódot (`err`). Amennyiben EACCES (engedély megtagadva) vagy ENOENT (nincs ilyen nevű fájl vagy könyvtár) állapotú és a könyvtár tulajdonosa (`dir->i_uid`) különbözik a pillanatnyi folyamatot futtató felhasználótól (`current ->fsuid`), de a jelenlegi felhasználó nem a rendszergazda (hiszen a rendszergazda bármit megtehet), akkor üzenetet jelenít meg, amiben megjelöli a kukkoló felhasználó azonosítóját. Az ilyen üzenetek általában a `syslogd`-be íródnak.

Titkosítás

Az eltérítő (wrapper) módszerek különösen alkalmasak a biztonsággal kapcsolatos alkalmazások fejlesztésére, ahol gyakran sokat segíthet az eltérítés vagy a megfigyelés. Nem meglepő, hogy a FiST legnépszerűbb alkalmazásai éppen a titkosított fájlrendszerek. Ebben a példában egy egyszerű, ***rot13*** kódolást alkalmazó titkosító fájlrendszert mutatunk be.
Rendszerünkön minden adatot a (feltételezhetően már korábban elkészült) ***rot13*** nevű algoritmussal szeretnénk titkosítani, ami értékként a bemenő és kimenő verem címét, valamint azok hosszát kapja meg. A korábbi példáinkkal ellentétben azonban most nincsen olyan nevezetes függvénymutató, ahová a `rot13()` függvényünket egyszerűen behelyezve elkódolhatnánk a fájlokat. Valójában bármely fájlrendszeren legyünk is, a fájlok adataival dolgozni meglehetősen összetett dolog, hiszen több függvénymutatóra is ügyelnünk kell, valamint a fájl adatai kétféleképpen, olvasási és írási rendszerhívásokkal is elérhetők, továbbá mindennek bármilyen fájleltolással, valamint az egész lapokon dolgozó `mmap`-pel is együtt kell működnie. A WrapFS, hogy megkönnyítse a halmozható fájlrendszer-fejlesztők életét, a fent felsorolt valamennyi függvénymutatót két egyszerű függvényben foglalja össze: az egyikben kódoljuk a fájladatokat, a másikban visszafejtjük őket, feltételezve, hogy egész laphatáron kezdődő adatokat használnak (például az IA-32 rendszereken ez 4 KB-ot jelent). A WrapFS-mintát használva az egyetlen dolog, amit nekünk kell megírnunk, a ***rot13*** alapú titkosító fájlrendszer. (A példa a 48. CD Magazin/FiST/1lista.txt-ben található.)
A WrapFS már eleve tartalmazza a kevert olvasásokat, írásokat és a memóriatérképes műveleteket kezelni képes összes bonyolult kódot. A WrapFS az `encode_block` segítségével kódolja az adatlapokat, illetve a `decode_block` alkalmazásával visszakódolja őket (példánkban e kettő azonos).
A ***rot13*** nem éppen legcélszerűbb kódolás, de az egyszerű példa alapján sokkal erősebb titkosítást használó fájlrendszereket is építhetünk. Ezt követtük mi is, amikor nemrégiben felépítettünk egy NCryptFS nevű, igen hatékony titkosító fájlrendszert (a CryptFS utódját). Az NCryptFS többféle kódolást is ismer; egy felhasználó, folyamat vagy csoport több kulcsot is használhat; többféle azonosítási módszere van; a kulcsok ideje lejárhat és visszavonhatók; kezeli az átruházott előjogokat; és még sok más képességgel is rendelkezik – mindezt elhanyagolható teljesítménycsökkenés árán.
A WrapFS a fájlnevek módosítását is kezeli két további eljárással, amik a fájlneveket kódolják és dekódolják. Mindenképpen ügyelnünk kell arra a fájlnevek kódolásakor, hogy a fájlneveknek a kódolás után is szabályosak maradjanak. Más szavakkal, nem tartalmazhatnak `null` értékeket vagy / (perjel) karaktert. Általánosan elterjedt megoldás, hogy titkosítás után `uuencode`-oljuk a fájlneveket.

Új képességek felhasználói alkalmazásokhoz

A `wrapfs_unlink` példánkban azt javasoltuk, hogy a fájlt törlése helyett egyszerűen csak nevezzük át, így mentve minden törölt állomány egy másolatát. Tegyük fel, hogy rendszerünket unrmFS-nek nevezik, ahol a törlendő fájlokat törlés helyett inkább eredeti F nevükről ***F.unrm*** alakúra nevezzük át. Elég bosszantó lenne, ha az összes ***.unrm fájl*** megjelenne a könyvtárunkban, főként ha ott egyáltalán nem is számítunk fájlokra. Továbbá ezekkel a szolgáltatásokkal akár a támadót is becsaphatjuk, aki megpróbálja letörölni a naplófájlokat, hogy eltüntesse a nyomait. Mindehhez azonban a ***.unrm*** fájloknak alapértelmezés szerint láthatatlanoknak és elérhetetlennek kell lenniük a felhasználó számára.
Ha bizonyos fájlokat el akarunk rejteni fájlrendszeren, két dolgot kell megtennünk. Először is meg kell akadályoznunk, hogy a fájl megjelenjen a `->readdir()` hívásokban. Ezt úgy tehetjük meg, hogy a `wrapfs_filldir` függvénybe olyan kódot illesztünk, ami az összes `->filldir()` híváshoz kerülő fájlnevet ellenőrzi, és `NULL` értéket ad vissza azokra a fájlokra, amiket nem szeretnénk megjeleníteni. Másodszor, meg kell akadályoznunk, hogy a felhasználók közvetlenül rákeressenek az állományra. Ezt úgy érhetjük el, hogy a `wrapfs_lookup` elején kikeressük a ***.unrm*** fájlokat.
Természetesen, ha ezeket a fájlokat mindenki elől elrejtjük, abból sok hasznunk nem származik. A jogosult felhasználóknak – bizonyos körülmények között – el kell tudniuk érni a fájlokat. Egyszerű megoldás például, ha a hívófolyamat UID-jét ellenőrizzük, és csak bizonyos felhasználók elől rejtjük el a ***.unrm*** állományokat. Ennél jobb megoldás, ha minden rendszerhívások anyját, az `ioctl` hívást alkalmazzuk. A WrapFS rendszerhez tetszés szerinti számban határozhatunk meg új `ioctl` hívásokat, hogy azután apró felhasználói programokkal kiaknázhassuk őket. Ezt a módszert használtuk például a titkosító fájlrendszerünkben, ahol a felhasználó titkosító kódját egy felhasználói szintű eszköz adja át a rendszermagnak.
Az unrmFS rendszerünkben tehát készíthetünk egy visszaállító `ioctl` hívást, ami a visszaállítandó F fájlnevet várja bemenetként, és ellenőrzi, hogy létezik-e ***F.unrm*** nevű állomány. Amennyiben igen, az ***F.unrm*** fájlt visszanevezi F-re, ismét megjelenítve az unrmFS-ben. (A példa a 48. CD Magazin/FiST/2lista.txt-ben található.)
A FiST programot, a leírást és a rengeteg további példát a ➲ http://www.cs.sunysb.edu/~ezk/research/fist címen találjuk.
Jó halmozást!

Linux Journal 2003. május, 109. szám

Erez Zadok (ezk@cs.stonybrook.edu)
A Stony Brook Egyetem Számítástechnika karán dolgozik, a Linux NFS and Automounter Administration szerzője, a FiST halmozható mintarendszer megalkotója, és az Am-utils önműködő befűzőrendszer első számú karbantartója.

Linux a gyakorlatban: mikrovezérlő-programozás

A Linux kapcsán legtöbbször vagy a számítógépes hálózati kiszolgálókról, vagy a felhasználói programokról esik szó.

Az a tény, hogy a számítógép elsősorban eszköz, eléggé háttérbe szorul. Az alábbiakban egy gyakorlati feladat megoldásának ürügyén mutatom be a nyílt forrású programok használatát: egy PIC mikrovezérlő program megírását, kipróbálását és eszközbe töltését követhetjük figyelemmel.

A feladat és az eszköz

Munkám során egy irányítástechnikai feladat – adatgyűjtés, egyszerű folyamatvezérlés – merült fel. Általában a Microchip által gyártott PIC mikrovezérlőket szoktam alkalmazni, mivel sokféle változat közül könnyen lehet az adott feladathoz illőt találni. Az újakban ráadásul flashmemória van, ami lehetővé teszi a törlés nélküli sokszoros (legalább ezer) újraprogramozást, így könnyebb a programfejlesztés, mert az alakuló programot magában az eszközben is ki lehet próbálni, nem csak a szimulátorban.
Mivel kis sorozatú vagy egyedi készülék esetén a lapkák közötti kisebb árkülönbség nem játszik jelentős szerepet, az egyik legnagyobb tudású mikrovezérlőt választottam, a 16F877-et.
A Microchip cég oldaláról le lehet tölteni egy fejlesztőrendszert az általa gyártott mikrovezérlőkhöz. Ez tartalmazza az assemblert, a különböző általuk gyártott vagy ajánlott programozó készülékek illesztőfelületét, valamint az összes vezérlőhöz szimulátort. Emellett a honlapjukon minden eszköz adatlapja (pdf formátumban), példaprogramok és tanulmányok is fellelhetők. Ugyanakkor két hátrányt meg kell említenem: a programok csak Windows alatt működő változatban léteznek, valamint az assembleren kívül más programnyelv csak külön díj ellenében szerezhető be.
A fentiek miatt elhatároztam, hogy megpróbálok találni valamilyen programot, eszközt, ami lehetővé teszi a könnyebb programfejlesztést, esetleg Linux alatt is.

A gputils programok és a gpsim

A keresés nem várt gyors eredménnyel járt, ugyanis a ➲ http://www.gnupic.org címen ***Scott Dattalo*** igen jó, PIC mikrovezérlőkkel kapcsolatos címlistájára akadtam rá. Ezt az oldalt használhatjuk kiindulópontként a programok letöltésekor. A legalapvetőbb programok a `gputils` csomagban találhatók. Mint kiderült, ezeknek a programoknak az egyik fő fejlesztője szintén Scott Dattalo. A következő főbb programok tartoznak hozzá: a `gpasm`, az assembler-fordító, `gplink` és `gplib`, amik az áthelyezhető kódként fordított programok könyvtárakba való szervezését, majd futtathatóvá szerkesztését teszik lehetővé. A fejlesztők célja az, hogy a gpasm fordítóforráskód szintjén teljes egészében helyettesíteni tudja az eredeti Microchip programot, az `mpasm`-t. Mivel a forráskód szabad, tetszőleges rendszerre lefordítható, amit Linux, Mac és Windows esetén már meg is tettek. Eddig eljutva jó érzéssel töltött el, hogy Linux alatti programfejlesztéshez is léteznek programok, azonban a fent vázolt célhoz nemigen jutottam közelebb. Szerencsére mindjárt a gputils honlapján megtaláltam (amúgy a gnupic oldalon is elérhető) a következő lépéshez vezető utat: a gpsim honlapjának a hivatkozását. Ez tulajdonképpen nem része a gputils csomagnak, de mivel ezt is Scott fejlesztette, nem csoda, hogy szervesen illeszkedik a gpsimhez. A célja az, hogy más fordítóprogramokkal és más mikrovezérlőkkel is együtt tudjon működni, de egyelőre a PIC-változat a legkidolgozottabb. Ezekből a legtöbb gyakran használt típust tudja szimulálni, nagyrészt kifogástalanul, az összes külső eszközzel együtt.

Néhány szó a gpsimről

A `gdb` hibakeresőhöz hasonlóan alapvetően parancssori működésű, azonban a gtk-extra csomagra épülve grafikus felületet készítettek hozzá (ha ezt akarjuk használni, X11 alatt indítsuk a programot). Az alapvető hibakereső szolgáltatások megtalálhatóak benne: léptetés, függvényátugrás, töréspont és futtatás. A mikrovezérlő tulajdonságainak megfelelően megtekinthetjük a programmemória, az adatmemória, vagyis a regiszterek (mivel ezek a vezérlők Harvard felépítésű RISC processzorokat tartalmaznak, így a kettő teljesen különálló, sőt ebben az esetben bitszélességük sem azonos), az esetleges belső EEPROM memória, valamint az áramkör lábainak értékét, illetve állapotát. Emellett külön állományban meg lehet adni az úgynevezett stimulusokat, amik a külső világot hívatottak szimulálni a mikrovezérlő számára. Ilyenek például az adatok a soros vonalon, a bemenetek változása, a számláló impulzus. A lefuttatott szimulációs ciklusról részletes nyomkövető állományt tud előállítani, amit utólag (mintegy offline) tanulmányozni lehet. Ezzel a programok fordítása és hibakeresése tekintetében gyakorlatilag elértem azt a szintet, amit az eredeti Microchip programokkal, és mindezt Linux alatt. Azonban még nem találtam megoldást a program eszközbe (lapkába) történő írására. Nos, ilyenre is rá lehet akadni a gnupic oldalon.

A beprogramozó

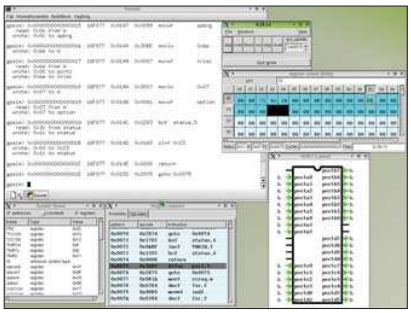
Többféle megoldást is fellelhető: saját építésű programozó (nyomtatókapus és USB-változat), valamint nyomtatókapus gyári programozókhoz való program, és nagy örömömre található Linux program a Microchip Picstart Plus soros illesztésű programozójához is. Annak idején vásároltam is egy ilyen programozókészüléket az alábbi előnyök kedvéért:

- viszonylag olcsó,
- szervesen illeszkedik a Microchip fejlesztőrendszerbe (ez Linux alatt már nem játszik szerepet),
- a Microchip honlapjáról mindig letölthető a legújabb gyári program (firmware), amit egy 17C44 vezérlőbe beprogramozva és ezt a készülékbe helyezve a legújabb gyártmányú lapkákhoz is mindig használható marad.

Korábban bosszankodtam, hogy a cég, jóllehet más tekintetben nagyon megbízható és a kisfelhasználóknak is minden útmutatást megad, a Picstart protokollját nem hozza nyilvánosságra, így linuxos meghajtóprogramját nem tudtam (és eddig más se) megírni. Tulajdonképpen nyomós oka van annak, hogy a cég ezt miért nem teszi meg: így nincs

```
include libnibble.fs  \ karakter --> hexa-szÆmjegyÆtalak t
include piceeprom.fs  \ eeprom mem ria rÆs, -olvasÆs
include picflash.fs       \ programmem ria-kezelØs
include libstrings.fs     \ sz vegek tÆrolÆsa Øs kezelØse

macro

: transmit-byte ( byte -- )
    begin txif bit-clr? while repeat
    txreg !
;

: transmit-nibble ( nibble -- ) nibble>hex transmit-byte ;

target

variable recv-state
variable recv-char
variable recv-dat

: receive-command ( -- )
    begin
        begin rcif bit-clr? while  repeat
        \ '!' vagy '?' parancsot vezet be, a t bbi
          ➥karaktert egyszerßen visszhangozzuk
        rcreg @ dup dup transmit-byte
        [char] ? = if drop 1 recv-state ! recurse exit then
        dup
        [char] ! = if drop 11 recv-state ! recurse exit then
        hex>nibble
        \ elsı adat karakter
        recv-state @ 1 = if
            swapf-tos recv-char ! 2 recv-state ! recurse
            ➥exit then
        \ mÆsodik adat karakter
        recv-state @ 2 = if
         \ olvasÆs
            recv-char @ or ee@ dup [char] = transmit-byte
            ➥4 rshift transmit-nibble 0f and
            ➥transmit-nibble 0 recv-state !
            $d transmit-byte recurse exit then
          \ elsı c m karakter
        recv-state @ 11 = if
            swapf-tos recv-char ! 12 recv-state ! recurse
            ➥exit then
          \ mÆsodik c m karakter
        recv-state @ 12 = if
            recv-char @ or recv-char ! 13 recv-state !
[char] = transmit-byte recurse exit then
          \ elsı adat karakter
        recv-state @ 13 = if
            swapf-tos recv-dat ! 14 recv-state ! recurse
            ➥exit then
         \ mÆsodik adat karakter
        recv-state @ 14 = if
         \  rÆs
            recv-dat @ or recv-char @ ee! 0 recv-state !
            ➥$d transmit-byte recurse exit then
        drop
    again
;
```

a lista folytatását lásd a következő oldalon



A gpsim futás közben

megkötve a keze, és a különböző gyári változatoknak nem kell egymással felcserélhetőnek maradniuk: hiszen mindig csak a pillanatnyi `mplab`, illetve `mpasm`-változathoz kell illeszkedniük. Nos, ***Andrew Pines*** most (jórészt) megoldotta a feladatot, ugyanis egy különleges soros kábelt készített, és miközben Windowsból használta a programozót, rákötött linuxos gépével mentette az adatokat, és ezek révén nagyon sok részt megfejtett a protokollból. Jelenleg adott egy olyan működő program, amellyel a legtöbb programbetöltési feladat megoldható. Olykor előfordulnak hibák, de azért használható.

A program lefordítása egyszerű: használatakor meg kell adnunk a soros kaput, amihez a programozó csatlakoztatva van. Ügyeljünk arra, hogy semmilyen eszköz ne használja ezt. (Azért tudom ilyen pontosan, mert mint kiderült, én elfelejtettem, hogy az egyik soros kapun a modem, a másikon egy soros terminál volt beállítva. Igaz, hogy régóta nem használtam egyiket sem, de a megfelelő programok futottak, és bekavartak az eseményekbe). A programozóban legalább a 3.00.04-es gyári program szerepeljen, ha ennél régebbi van, frissíteni kell (a Chipcad Kft. honlapján akár a felprogramozott 17C44 is megrendelhető). Ha minden össze van kötve, akkor a

```
picp /dev/ttyS0 16f877 -v
```

parancsra kiírja a gyári program változatszámát. A használt soros kapunak és a programozóban lévő vezérlőnek megfelelően módosítani kell a parancsot. Lehetőség nyílik az eszköz programmemóriájának és beállítási szavának olvasására és írására (`-wp`, illetve `-rp` vagy `-wc` és `-rc`), a nem flasheszközök üres állapotának ellenőrzésére és a flasheszközök törlésére. Egyelőre az EEPROM adatmemória írása és olvasása nincs megvalósítva. Az „adatlopó” program is benne rejlik a forrástárban, továb-

a lista folytatása az előző oldalról

```
: print ( -- ) begin str-char dup while transmit-byte
➥repeat drop ;
: nyito ( -- ) c" PIC soros teszt v0.1 (c)
➥Havranek Ferenc " print $d transmit-byte ;

: init ( -- )
    $06 adcon1 !          \ A/D Ætalak t  tiltÆsa
    $ff trisa !           \ porta bemenet
    0 trisb !             \ portb kimenet
    0 portb !             \ alacsony szint (ki)
    $90 rcsta !           \ RS232 engedØlyezØse, folyamatos
                            ➥fogadÆs
    $24 txsta !           \ nagysebessØgß bitrÆta-lØtrehoz
    $0c spbrg !           \ 19200 baud 4 MHz-nØl
    $be trisc !           \ C6 kapu TX
    $07 option_reg !
    0 recv-state !        \ kiindul  Ællapot

;

main : main ( -- ) init nyito receive-command ;

\ LapkabeÆll tÆsok

fosc-hs set-fosc
false set-boden
false set-wdte
```

bá a megfelelő kábel leírása is, így akár magunk is megpróbálkozhatunk az utólagos felderítéssel.
Ezzel már valóban eljutottunk arra a szintre, hogy majdnem minden olyat meg tudunk tenni, amit az eredeti fejlesztőeszközökkel. De a célkitűzésem az volt, hogy valamilyen, a programfejlesztést jelentősen megkönnyítő eszközt is keressek. Nos, hála ***Samuel Tardieu***-nek, találtam is ilyet.

A forth fordító

Mielőtt (röviden) ismertetném a programot, néhány szó a forth nyelvről, mivel az utóbbi időben – megítélésem szerint – érdemtelenül mellőzve van.
A forth egy láncolt kódot előállító értelmező nyelv. Legalábbis az volt, amikor ***Charles Henry Moore*** egy csillagvizsgáló teleszkóp irányításának programozását megkönnyítendő megalkotta az első megvalósítását. Ő egyébként azóta is lelkes híve maradt ennek az ötletnek: már több ilyen elven működő processzort készített, részben a saját maga által alkotott (és szintén forthban írt) áramkörtervező programmal. Érdekességként: az általa alkotott processzor az adott feladatot végrehajtó program futtatása esetén azonos teljesítményt nyújtott, mint egy (a szóban forgó időpontban korszerű) 486-os processzorú gép, miközben a megvalósított tranzisztorszolgáltatások száma a lapkán kevesebb mint annak egytizede volt. A forth nyelv fő jellemzője, hogy úgynevezett szavakból áll, valamint az adatokat egy veremtáron keresztül továbbítja közöttük. Ami igazán széppé teszi, az az, hogy tetszőleges szavakat adhatunk hozzá, és a továbbiakban ugyanúgy használhatjuk őket, mint a nyelv alapszavait (a C-függvényekhez hasonlóan). Mivel a forth értelmező- és fordítóprogramok általában láncolt kódot állítanak elő, rendkívül kis méretű, elfogadható sebességű programot kapunk eredményül. Az ügyesen megírt programok még olvashatók is.
Az általam megtalált program a picforth, tulajdonképpen nem is program a szó ma használatos értelmében, azaz nem önállóan végrehajtható. Inkább parancsfájlnak nevezhetnénk, mivel a futásához szüksége van egy, a futtató gépre készült forth-értelmezőre. Ebben az esetben erre a célra a gforth (GNU forth) használható, ami az ANS forth szabvány egy nyílt forrású megvalósítása.
A program lényegi része a ***picforth.fs*** fájlban bújik meg. Emellett létezik még néhány segédfájl, amikben a mikrovezérlő EEPROM és programmemóriájának írását, illetve olvasását megkönnyítő szavak találhatók, valamint táblázatok és szövegek tárolását és előhívását lehetővé tevő segédletek. Ezenkívül egy rövid leírás mellett több, a forth esetén különösen fontos, működő (azaz kipróbált) példaprogram. A `Makefile` jól használható, végeredményként a picp programmal beégethető ***.hex***, az assembler listát tartalmazó ***.disasm*** és a szavak címeit tartalmazó ***.map*** állományt állít elő.
A ***.hex*** fájl a gpsim programba betölthető (előbb ki kell adni benne a `processor pic16f877` parancsot), és ki is próbálható. Ez látható a képernyőn is az alábbi program (egy soros vonalon keresztül a vezérlő EEPROM memóriájának írását és olvasását egyszerű parancsok segítségével megvalósító) lefordítása után, lásd *listánkon*.
Már csak egyetlen dolog hiányzik: hogy ki tudjuk próbálni a programot. Ehhez természetesen szükség van egy PIC kártyára, 16f877-tel, 4 MHz-es oszcillátorral és megvalósított soros (RS232) ki- és bemenettel. Ezenkívül szükségünk lesz valamire, ami nyers adatot tud küldeni és fogadni (megjeleníteni) a soros vonalon. Némi keresgélés után kiderült, hogy a Linux Programmers Guide példaprogramjai között akad egy egyszerű soros terminálszimulátor program: a miniterm. Ennek forrásába be kell írni a soros kapun kívánt értékeket (melyik például ttyS0, vagy a sebesség: 19200 baud), majd le kell fordítani, és már futhat is!
Mint látjuk, ismét kiderült, hogy a Linux szinte minden feladat megoldására alkalmas, valamint arra is fény derült, hogy legyen bármilyen a program- és eszközeszményképünk, a világban biztosan találunk olyan embereket, akik hozzánk hasonlóan gondolkodnak, hasonló megoldásokat keresnek, és sok esetben munkájuk eredményét közkinccsé is teszik.
Ha valakit érdekel a mikrovezérlő, akár kedvtelésből, egy későbbi cikkemben ismertetek egy olyan kártyatervező és szerkesztőprogramot, amivel a vezérlőnkhöz elkészíthetjük a nyomtatott áramkör tervét. Vajon hol is alkalmazhatjuk őket? Lássuk csak: hőmérsékletmérésre lakásban, udvaron, számítógépházban, de akár lakásautomatizálásra is.

Havránek Ferenc
Automatikamérnökként dolgozik. Kedvtelései közé tartozik mindenféle kétkerekű járművön való közlekedés. Szívesen tölti idejét programozással, nemcsak PC-s, hanem egyéb környezetben.

Ismerjük meg a szabályos kifejezéseket!

A szövegfeldolgozás és szövegleírás terén aligha találunk a szabályos kifejezéseknél hatékonyabb eszközt.

Képzeljük el, hogy ki szeretnénk keresni egy nevet a telefonkönyvből, de nem emlékszünk a név pontos írásmódjára. Hosszú időt tölthetünk el az összes lehetséges név kikeresésével, hacsak nincs egy olyan eszközünk, amelyik kigyűjti azt a viszonylag kevés lehetőséget, ami a hiányos tudásunknak megfelelő keresési feltételre illeszkedik. A szabályos kifejezések pontosan ilyen eszközök.

A szabályos kifejezés egy olyan karakterlánc, ami egy másik karakterláncot vagy karakterlánccsoportot ír le. Számos alkalmazás aknázza ki ezt a lehetőséget, néhány ezek közül: Perl, sed, awk, egrep, sőt még az Emacs is (a cikk elolvasása után próbáljuk ki a Ctrl-Alt-% billentyűkombinációt).

Bizonyosfajta szabályos kifejezéseket valószínűleg mindannyian használtunk már. Az `ls *.pl` héjparancsban a ***.pl*** egy olyan szabályos kifejezés, ami a következő karakterláncokat írja le: tetszőleges számú tetszőleges karakter (*), utána egy pont (.), végül két adott karakter (pl).

A szabályos kifejezések előállításának szabályai az összes elképzelhető karakterlánc leírását lehetővé teszik, attól függetlenül, hogy azok milyen összetettek. Sajnos a valóságban a helyzet ennél bonyolultabb, ugyanis a szabályos kifejezéseknek legalább kétféle változata létezik: kiterjesztett és egyszerű. Ráadásul nem minden alkalmazás támogatja az összes szabályt.

Bevezetés a szabályos kifejezések használatába

Akkor mondjuk, hogy egy szabályos kifejezés illeszkedik egy karakterláncra, ha azt helyesen írja le. Egy adott szabályos kifejezés akárhány karakterláncra illeszkedhet. Az a szokás, hogy a szabályos kifejezéseket törtjelek közé =(/.../) írjuk. A következőkben a kiterjesztett szabályos kifejezésekkel fogunk dolgozni.

A legegyszerűbb szabályos kifejezés csak betűkből és számokból áll, az ilyen kifejezés az olyan összes karakterláncra illeszkedik, ami tartalmazza őt részkarakterláncként. Vegyük például a következő részletet a kedvenc Rossini-operámból: „Zitto, zitto, piano, piano, senza strepito e rumore.” A `/piano/` szabályos kifejezés illeszkedik a szövegre, mert ugyanazokat a karaktereket ugyanabban a sorrendben tartalmazza, mint amit a szabályos kifejezésben megadtunk.

A jobb megértés kedvéért játszhatunk a következő Perl-parancsfájllal. Változtassuk meg néhányszor a szabályos kifejezést:

```
#!/usr/bin/perl
$verse = "Zitto, zitto, piano, piano, senza "
.          "strepito e rumore";
if ($verse =~ /piano/) {
   print "Illeszkedik!\n";
} else {
   print "Nem illeszkedik!\n";
}
```

A Perl nyelvben az =~ műveleti jel két szabályos kifejezést hasonlít össze, és ha illeszkedést talál, „igaz” értéket ad vissza. Néhány karakter (a nevük metakarakter) nem egyszerű karakternek számít, hanem különleges célokra van fenntartva. Például a * (csillag) arra használatos, hogy egy karaktercsoport nulla vagy több előfordulására illeszkedjen. A karaktercsoportot, más néven atomot olyan módon adjuk meg, hogy az egyetlen egységnek kezelt karaktereket zárójelek közé zárjuk. A `/(piano,)*` szabályos kifejezés illeszkedik a példaszövegre, mert az atomot alkotó „piano,” karaktersorozat kétszer ismétlődik. Ha az atom csak egyetlen karakterből áll, akkor a zárójel elhagyható.

A * karakter jelentése a szabályos kifejezésekben eltér a héjbeli jelentésétől. A szabályos kifejezésekben a * módosító; a bal oldalán elhelyezkedő atom többszöri előfordulását írja le. Emiatt a „piano” karakterláncra illeszkedik a `p*` a héjban, de a `/p*/` szabályos kifejezés csak a p, pp, ppp-re, illetve az üres karakterláncra illeszkedik.

Az atom N és M közötti számú előfordulásának megadásához a {N,M} jelet használhatjuk. A {N} olyan karakterláncokra illeszkedik, amelyek az atom pontosan N darab ismétlődését tartalmazzák. A {N,} a legalább N darab ismétlődésre illeszkedik. A következő szabályos kifejezések illeszkednek:

```
/( piano,){0,10}/
/( piano,){1,2}/
/( piano,){2}/
```

Az első szabályos kifejezés természetesen illeszkedne a „piano, piano, piano” karakterláncra is. A + és a ? metakarakterek a {1,} és a {0,1} rövidítései.

Az illeszkedő zárójeles atomok különleges változókban tárolódnak, amiket a \ (per)jelet követő szám azonosít. A szabályos kifejezésben előforduló első zárójeles atom a `\1` változóban, a második a `\2` változóban tárolódik. Például a

```
/Z(itto), z\1, ( piano,)\2/
```

illeszkedik a fent megadott idézetre (`\1 = "itto"* Øs \2 **= "piano,"`).

A . (pont) metakarakter bármelyik karakterre illeszkedik, emiatt a `/.(itto),.\1/` szabályos kifejezés illeszkedik a „Zitto, zitto” és a „zitto, zitto” karakterláncra is. Ezenkívül illeszkedik még például a „Ritto, ritto” karakterláncra is, ami már nem ugyanazt jelenti. Ha a túlzott általánosítást el szeretnénk kerülni, a lehetséges karaktereket szögletes zárójelek között megadhatjuk `/[Zz](itto), [Zz]\1/`.

A kötőjel a szögletes zárójelek között karaktertartományt jelöl. Például a `/[a-z]/` az összes kisbetűre, a `/[A-Z]/` pedig az összes nagybetűre illeszkedik. A `/[a-zA-Z0-9_]/` minden betű, szám és aláhúzásjel karakterből felépülő karakterláncra illeszkedik.

A | (cső) metakarakter a különböző lehetőségek megadására szolgál. Működése hasonló a logikai VAGY művelethez. Emiatt a `/Zitto|zitto/` illeszkedik a „Zitto” és a „zitto” karakterláncra is.

A ^ és a $ (dollárjel) metakarakterek a karakterlánc elejére és végére illeszkednek. A szögletes zárójelen belül a ^ jel a tagadás művelete. Emiatt a `/[^a-z]itto/` illeszkedik a „Zitto” karakterláncra, de a „zitto” karakterláncra nem, mert a `[^a-z]` jelentése: „tetszőleges betű, ami nem kisbetű”. Ha a metakaraktert közönséges karakterként szeretnénk értelmeztetni, tegyünk elé egy fordított perjelet (\). Ekkor a szabályos kifejezést értelmező program tudni fogja, hogy ezt egyszerű karakterként kell értelmeznie.

A szabályos kifejezések használata

Értékelni fogjuk a szabályos kifejezés erejét, ha megvizsgáljuk a következő kis Perl-parancsfájlt, ami segít a rendszergazdáknak kiszűrni a sikertelen bejelentkezési kísérleteket.
A következő példákban szemléletes szabályos kifejezéseket fogok használni a különböző lehetőségek bemutatása céljából. Ugyanezeket a feladatokat egyszerűbb szabályos kifejezésekkel is megvalósíthatjuk.
Ha valakinek sikertelen a bejelentkezési kísérlete, a `syslogd` az alábbihoz hasonló üzenetet ír a */var/log/messaged* naplóállományba:

```
Jul 26 16:35:25 myhost su(pam_unix)[2549]:
➥authentication failure; logname=verdi uid=500
➥euid=0 tty= ruser=organtin rhost=  user=root
Jul 27 14:54:36 myhost login(pam_unix)[688]:
➥authentication failure; logname=LOGIN uid=0
➥euid=0 tty=tty1 ruser= rhost=  user=mozart
```

A bejegyzésből kiolvasható a bejelentkezési kísérlet időpontja, a felhasználó neve, aki egy másik felhasználó nevében be akart lépni – amennyiben ez az adat rendelkezésre áll –, és a célfelhasználó. Például a verdi felhasználó kétszer be akart lépni a rendszergazdaként, míg valaki mozart felhasználónévvel szeretett volna belépni a konzolról.
Vessünk egy pillantást az 1. listán (48. CD Magazin/Szabalyos könyvtár) bemutatott Perl-parancsfájlra: az a feladata, hogy beolvassa a */var/log/messages* állományt, azonosítsa az érdekesnek tűnő sorokat, és csak a lényeges adatokat gyűjtse ki.
Először is egyedül azokat a lényegi sorokat választjuk ki, amelyek illeszkednek az `/authentication failure/` szabályos kifejezésre (7. sor), minden egyebet eldobunk. Ezután minden sort egy olyan szabályos kifejezéssel kezelünk (8. sor), ami a következő karakterláncra illeszkedik: a karakterlánc eleje (^) pontosan három ({3}) betűből áll (`[a-zA-Z]`), ezt egy szóköz követi, majd legfeljebb két (+) karakter kerül sorra, amit számok (0–9, amit a Perlben a `\d` metakarakterrel is jelölhetnénk) vagy egy szóköz követ. A szóköz után tetszőleges számú (*) számjegy és pontosvessző lehet. A karakterlánc eddig leírt része zárójelben van, tehát a `\1` változóba kerül. Ezután tetszőleges számú karakter (.*) következhet a `"logname="` karakterlánc előtt. Ezt a karakterláncot tetszőleges számú betű vagy számjegy követheti. A zárójelpár következtében ez a `\2` változóban tárolódik. Végül tetszőleges számú karakter lehet a `"user="` karakterlánc előtt, amit tetszőleges számú betű és számjegy követhet. Ez az egész a `\3` változóba kerül.
Ebből a példából megtanulhatjuk, hogyan lehet részkarakterláncokat kivenni a karakterláncokból. Nem kell ismernünk a részkarakterláncok viszonylagos helyzetét, ha le tudjuk írni a kinézetüket.
A Perl hasznos szolgáltatása, hogy a szabályos kifejezések változóiból önműködően Perl-változókat hoz létre, amik a `$1`, `$2` stb. neveken érhetők el, miután a szabályos kifejezés illesztése megtörtént. A Perl hasznos jelek használatát is megengedi az ugyanazt jelentő Posix-megfelelő jelek mellett, ilyen például a már említett `\d` és a `\w`, ami megfelel a `[A-Za-z0-9_]` kifejezésnek. További tudnivalók a Perlről a súgóoldalon (man perlre) olvashatók.

Egyszerű szabályos kifejezések

Az egyszerű szabályos kifejezéseket számos más program is használja, például a sed vagy az egrep.
Az egyszerű szabályos kifejezésekben a |, a + és ? metakarakterek nem léteznek, és a gömbölyű, illetve kapcsos zárójeleket védeni kell, ha metakarakterként szeretnénk használni őket. A ^, a $ és a * metakarakterekre bonyolultabb szabály vonatkozik (további részletekért lásd: man 7 regex). A legtöbb esetben azonban kiterjesztett társaikhoz hasonlóan viselkednek. Gyakran a szabályos kifejezést a kiterjesztett alakban kényelmes szerkeszteni, és szükség esetén adni hozzá a védőkaraktereket.
A 2. listában (48. CD Magazin/Szabalyos könyvtár) bemutatott példa böngészőprogramban megtekinthető, html formátumú oldalt hoz létre a rendszernapló állományából. A HTML-elemek kiírása mellett, amelyek az oldal fejlécét és a táblázatot hozzák létre, kilistázzuk egy adott könyvtár összes állományát, és az eredményt a sed programba vezetjük, ami azt szabályos kifejezések segítségével átalakítja. A sed által is használt szöveghelyettesítés eléggé elterjedt:

```
s/kifejezØs/csere/
```

Ebben a kifejezés szabályos kifejezés, amit le kell cserélni.
A megadott szabályos kifejezés egy kilenc elemből álló karakterláncot ír le. Például a `[rwxds-]` az első elem lehetséges karaktereit adja meg. A karakterlánc ez utáni része betűket és számjegyeket tartalmaz, amiket perjelek tagolnak. Vegyük észre, hogy ebben az esetben a `(.*\/)(.*)` szabályos kifejezést használtuk. Az első csoport illeszkedik a perjelet megelőző összes karakterre, vagyis az elérési útra. A második csoport illeszkedik a többi karakterre (azaz a fájlnévre). Az elérési útban tetszőleges számú perjel lehet. A szabályos kifejezések (az egyszerűek és a kiterjesztettek is) mohók, azaz a lehető legtöbb karakterre próbálnak illeszkedni.
A parancsfájl eredménye a szabályos kimenetre íródik, amit egy adott állományba átirányíthatunk (például a `cron`-ból meghatározott időközönként), hogy megjelenjen a Weben.

Összegzés

A szabályos kifejezések messze a leghatékonyabb eszközök a szövegfeldolgozási és szövegleírási feladatokra, és Linux-környezetben számos alkalmazás támogatja őket. Sajnos a legnépszerűbb keresőmotorok (tudtommal) egyáltalán nem támogatják a szabályos kifejezéseket, éppen a bonyolultságuk miatt. Képzeljük csak el, milyen pontos lehetne a keresésünk, ha a keresett weboldalt szabályos kifejezéssel írnánk le!

Linux Journal 2003. május, 109. szám

Giovanni Organtini (g.organtini@roma1.infn.it)
A római egyetemen tartja a „Bevezetés a számítástechnikába és a programozásba fizikusok számára” című óráit. Évek óta használ Linuxot, szórakozáshoz és munkához egyaránt.

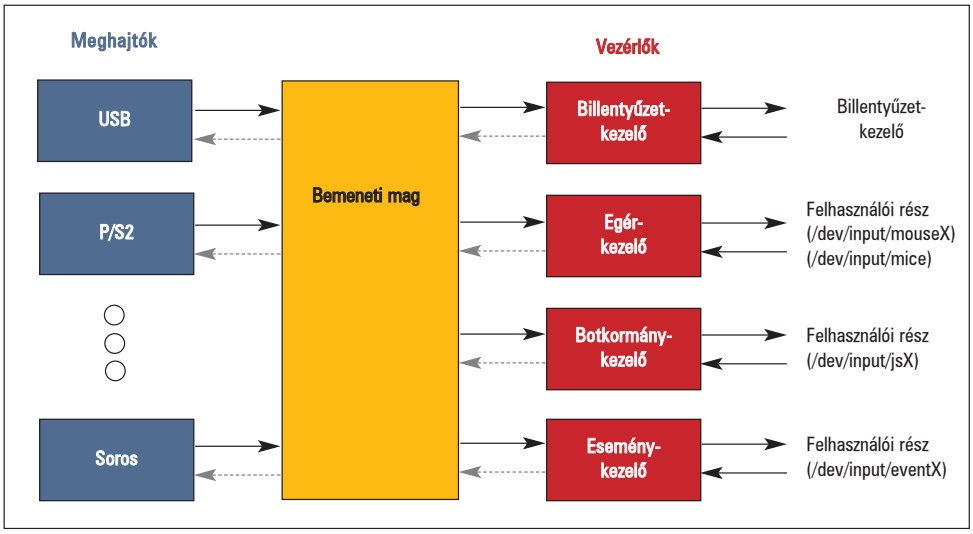
A linuxos USB alrendszer (1. rész)

Az USB bemeneti alrendszer egyre inkább terjed a magváltozatokkal. Épp itt az ideje megérteni, hogy pontosan mi is a dolga eszközeinkkel.

A linuxos USB alrendszer egyszerű, jól összehangolt módszer minden bemeneti eszköz kezelésére. A Linuxban ez egy viszonylag új megközelítés, ami részben megtalálható a rendszermag 2.4-es változataiban, és teljes egészében beépítették a rendszermag 2.5-ös fejlesztői változataiba. Ez a cikk négy alapterületet hivatott megismertetni: leírja, mit tesz pontosan a bemeneti alrendszer, bemutatja az alrendszer rövid történeti áttekintését, elmondja, hogyan épül fel a bemeneti alrendszer a rendszermagban, valamint bemutatja a felhasználói szintű API-t, vagyis az alkalmazásprogramozói felületet, és azt, hogy miként tudod ezt felhasználni programjaidban. Az első három témakörre még ebben a cikkben kitérünk, a felhasználói szintű API-t és a befejező témakört a következő rész tárgyalja.

Mi az a bemeneti alrendszer?

A bemeneti alrendszer a Linux-rendszermagnak az a része, ami a különböző bemeneti eszközöket kezeli (például a billentyűzetet, az egeret, a botkormányt, a digitáblát és sok más egyéb eszközt), ezek segítségével lép kapcsolatba a felhasználó a rendszermaggal, a parancssorral és a grafikus felülettel. Ezek az alrendszerek a rendszermagban találhatók, mivel elérésükhöz valamilyen különleges alkatrészcsatolót kell alkalmazni (mint például a soros kaput, a PS/2-es kaput, az Apple Desktop Bust és az Universal Serial Bust, vagyis az USB-t), amik nem érhetők el közvetlenül, és a rendszermag kezeli őket. Ezt követően a rendszermag egy általánosan összefüggő, az adott eszköznek megfelelő felületen keresztül teszi elérhetővé a kezelést a felhasználói programok számára, különböző programozói felületeken keresztül.



Kapcsolat a beviteli alrendszer részei között

Hogyan jutottunk el eddig?

A Linux bemeneti alrendszer elsősorban *Vojtech Pavlik* munkája, aki először a botkormányok támogatásának előkészítésekor érezte egy rugalmas bemeneti rendszer szükségességét, majd később is, amikor az USB-n kezdett el dolgozni. Az első sokrétű bemeneti alrendszer a már meglévő botkormány- és USB-meghajtókat támogatta a 2.3-as rendszermagok idején, ezt továbbvitték a 2.4-be, ahol ez a támogatás továbbra is a botkormányokra és az USB-re korlátozódik.

A 2.5-ös rendszermagban található bemeneti alrendszert már az összes bemeneti eszközre kiterjesztették. Cikkünk ezen az összetett rendszeren alapul, ami a 2.6-os rendszermagokban a bemeneti alrendszer programozói felülete lesz. Annak ellenére, hogy cikkünk írásakor a jelenlegi 2.4-es és 2.5-ös rendszermagok programozói felületei valamelyest eltérnek, folyamatos munka folyik, hogy ezek a különbségek megszűnjenek, amit többnyire a 2.4-es rendszermag frissítésével érnek el.

Pillantás a kulisszák mögé, avagy a rendszermag működésének vizsgálata

A bemeneti alrendszer három eleme: a bemeneti mag, az eszközmeghajtók és az eseménykezelők. A köztük lévő kapcsolat az *ábrán* látható. Amellett, hogy alapesetben az út az alacsony szintű eszközöktől az eszközmeghajtóig, az eszközmeghajtótól a bemeneti magig, a bemeneti magtól a kezelőig, a kezelőtől a felhasználói területre vezet, általában létezik valamilyen visszaút is. Ez a visszaút teszi lehetővé például a billentyűzet lámpáinak vezérlését és az erő-visszacsatolású (force-feedback) botkormányok mozgásának szabályozását. Mindkét irány ugyanazt az eseménymeghatározást alkalmazza, csak különböző típusú azonosítókkal.

A különböző részek együttműködését események vezérlik, amik szerkezetekként (structure) vannak meghatározva (*1. lista*). Az első mező, a ***time*** egy egyszerű időbélyeg, az ezt követő mezők már valamivel érdekesebbek. A ***type***, vagyis típus mező az esemény általános típusát jelöli, például, hogy az egéren nyomtunk-e le egy gombot vagy a billentyűzeten; de vonatkozhat a mozgásra is, mondjuk a botkormány esetében. A ***code*** mező mondja meg, hogy melyik gombot nyomták le pontosan, vagy azt, hogy az elmozdulás melyik tengely mentén történik; míg a ***value***, vagyis érték mező azt jelzi, hogy milyen a lenyo-

1. lista Az event-dev-struct.txt

```
struct input_dev {

    void *private;

    char *name;
    char *phys;
    char *uniq;
    struct input_id id;

    unsigned long evbit[NBITS(EV_MAX)];
    unsigned long keybit[NBITS(KEY_MAX)];
    unsigned long relbit[NBITS(REL_MAX)];
    unsigned long absbit[NBITS(ABS_MAX)];
    unsigned long mscbit[NBITS(MSC_MAX)];
    unsigned long ledbit[NBITS(LED_MAX)];
    unsigned long sndbit[NBITS(SND_MAX)];
    unsigned long ffbit[NBITS(FF_MAX)];
    int ff_effects_max;

    unsigned int keycodemax;
    unsigned int keycodesize;
    void *keycode;

    unsigned int repeat_key;
    struct timer_list timer;

    struct pm_dev *pm_dev;
    int state;

    int sync;

    int abs[ABS_MAX + 1];
    int rep[REP_MAX + 1];

    unsigned long key[NBITS(KEY_MAX)];
    unsigned long led[NBITS(LED_MAX)];
    unsigned long snd[NBITS(SND_MAX)];

    int absmax[ABS_MAX + 1];
    int absmin[ABS_MAX + 1];
    int absfuzz[ABS_MAX + 1];
    int absflat[ABS_MAX + 1];

    int (*open)(struct input_dev *dev);
    void (*close)(struct input_dev *dev);
    int (*accept)(struct input_dev *dev,
                  struct file *file);
    int (*flush)(struct input_dev *dev,
                 struct file *file);
    int (*event)(struct input_dev *dev,
                 unsigned int type,
                 unsigned int code,
                 int value);
    int (*upload_effect)(struct input_dev
         *dev, struct ff_effect *effect);
    int (*erase_effect)(struct input_dev
         *dev, int effect_id);

    struct list_head        h_list;
    struct list_head        node;
};
```

mott egérgomb vagy billentyű állapota, vagy azt, hogy milyen elmozdulásról van szó. Ha például a típus egérgomb vagy billentyű, a kód megmondja, hogy melyik egérgomb vagy billentyű az pontosan; az érték mező pedig elárulja, hogy a gombot éppen felengedték-e vagy pont lenyomták. Ehhez hasonlóan, ha a típus valamilyen viszonylagos tengelyt jelöl, a kód elárulja, hogy pontosan melyik tengelyről van szó, az érték mező pedig az elmozdulás nagyságát és irányát tartalmazza. Ha az egeret átlósan mozgatod, miközben ezzel egyidejűleg az egér görgőjét (scrolling-wheel) is mozgatod valamelyik ujjaddal, akkor minden frissítéskor három viszonylagos eseményt kapsz vissza: egyet a függőleges mozgásból (y tengely), egyet a vízszintes mozgásból (x tengely) és egyet a görgő mozgásából adódóan.
Az eseménykezelők szolgáltatnak felületet a felhasználói réteghez, olyan módon, hogy a szabványos eseményformátumokat úgy alakítják át, hogy azt a különböző programozói felületek megértsék. Általában ezek a kezelők gondoskodnak a különböző eszközcsomópontok (a */dev* bejegyzések) kezeléséről is. A legáltalánosabb kezelő a billentyűzet kezelője, ez az a szabványos bemenet, amivel a programozók (különösen a C-programozók) a leginkább tisztában vannak.
Az eszközmeghajtók általában az alacsony szintű alkatrészekkel tartják a kapcsolatot, mint például az USB-vel, a PCI-jal, a memóriával, a különböző ki- és bemeneti (I/O) területekkel és a soros kapu ki- és bemeneti területeivel. Ezek alakítják át a részegység által küldött felhasználói bemenetet az általános üzenet formájába, még mielőtt elküldik azt a bemeneti magnak. A bemeneti mag a rendszermag szabványos csatlakozóbővítmény (plugin) szerkezetét használja, az `input_register_device()` függvénnyel vesz fel új eszközöket, míg az `input_unregister_device()` függvénnyel távolít el meglévő eszközöket. Ezeket a függvényeket az `input_dev` kapcsolóval kell meghívni, ezt az első listában láthatjuk. Annak ellenére, hogy ez a szerkezet meglehetősen hosszúnak tűnik, a legtöbb bejegyzés arra szolgál, hogy az eszközmeghajtó beállíthassa, hogy az adott eszköz milyen lehetőségekkel bír, vagyis azt, hogy milyen típusú események és kódok fogadására vagy küldésére képes.
Azon túl, hogy a bemeneti mag kezeli az eszközmeghajtókat és a kezelőket, egy */proc* fájlrendszer felületet is létrehoz, amin keresztül látható, hogy éppen milyen eszközmeghajtók és kezelők működnek. Alább egy jellemző példa látható a */proc/bus/input/devices* fájlból, ami egy USB-s egér működését mutatja:

```
I: Bus=0003 Vendor=046d Product=c002
Version=0120
N: Name="Logitech USB-PS/2 Mouse M-BA47"
P: Phys=usb-00:01.2-2.2/input0
H: Handlers=mouse0 event2
B: EV=7
B: KEY=f0000 0 0 0 0 0 0 0 0
B: REL=103
```

Az I: sor tartalmazza az azonosító adatokat – a 3-as busztípus (ami az USB) és a gyártó, a termék és a változat adatait az egér USB-leíróiból. Az N: sor a nevet mutatja, amit úgyszintén az

USB eszközleírói tartalmaznak. A P: sor a fizikai eszköz adatait tartalmazza, ez esetünkben az USB-vezérlő PCI címéből, az USB-fából és a bemeneti eszközfájlból áll. Az `input0` azt jelöli, hogy ez az első logikai bemeneti eszköz az adott fizikai bemeneti eszközhöz. Néhány eszköz, mint például a multimédiás billentyűzetek, a fizikai eszköz egyes részeit rendelhetik az egyik logikai bemeneti eszközhöz, míg a fizikai eszköz más részeit egy másik bemeneti eszközhöz rendelhetik. A H: sor az eszközhöz rendelt kezelőt tartalmazza, erre később még kitérünk. Az egyes B: sorok a különböző bitmezőket takarják, amelyek az eszköz képességeit írják le – ebben az esetben az egyes egérgombokhoz tartozó billentyűket, a viszonylagos tengelyeket a golyóhoz és a görgőhöz.

A */proc* felület nagyon jól használható egyszerű eszközmeghajtók kipróbálására. Vegyük például azt az esetet, amelyben az eszközmeghajtó befűzéskor bejegyzi magát, illetve amikor az eszközt eltávolítjuk, az eszközmeghajtó is eltávolítja a hozzátartozó bejegyzést (*2. lista*). Ehhez néhány bevezető művelet elvégzésére van szükség az `init_input_dev()` függvénnyel. A függvény beállítja az eszköz nevét, a fizikai és azonosító leíróit, és beállítja a bittömböket, hogy jelezze: az eszköz képes bizonyos típusú események fogadására (az EV_KEY jelzi az egérgombokat és a billentyűket, két különböző kód a KEY_A és KEY_B jelzi a gombok címkéit). Az előkészítő eljárás ezt követően bejegyzi az eszközt a bemeneti magba. Ha ezt a kódot hozzáadod a rendszermaghoz (a `modprobe` paranccsal), látni fogod, hogy egy új eszköz jelenik meg a */proc/bus/input/devices* fájlban, mint az az alábbiakban is látható:

```
I: Bus=0019 Vendor=0001 Product=0001
Version=0100
N: Name="Example 1 device"
P: Phys=A/Fake/Path
H: Handlers=kbd event3
B: EV=3
B: KEY=10000 40000000
```

Ha valóban üzenetet szeretnénk küldeni az eszközmeghajtónktól a bemeneti magnak, előtte meg kell hívnunk az `input_event`-et vagy valamelyik kényelmes közvetítőt, például az `input_report_key`-t vagy az `input_report_abs`-t, amiket a ***linux/input.h***-ban találunk meg. A *3. listában* egy olyan kódot láthatunk, ami erre mutat példát. Ez a példa nagyjából ugyanazt mutatja, mint az előző, azzal a különbséggel, hogy itt egy időzítőt adunk hozzá, ami az `ex2_timeout()` függvényt hívja meg. Ez az új programrész négy KEY_A és négy KEY_B lenyomását továbbítja. A művelet hatására összes 16 esemény jön létre, mivel külön események keletkeznek a billentyűk lenyomásakor és elengedésekor is. Ezek az események továbbítódnak a bemeneti maghoz, azt követően a billentyűzetkezelőhöz, ami az „aaaabbbb” vagy az „AAAABBBB” betűsorok megjelenését eredményezni a kezelőfelületen vagy a parancssorban, a SHIFT billentyű állapotától függően. Az időzítő úgy állítódik be, hogy négy másodpercenként újra meghívódjon, egészen a végtelenségig. A négy másodperces várakozási idő azért lett beiktatva, hogy – ha eleged van a megjelenő karakterekből – el tudd távolítani a bővítményt. Ha csökkented a várakozási időt, győződj meg róla, hogy más módon is hozzá tudsz-e férni a rendszerhez, esetleg egy másik SSH-kapcsolaton keresztül. Ne felejtsd el meghívni az `input_sync` függvényt sem! Ez a függvény értesíti az eseménykezelőt (esetünkben a billentyűzetkezelőt), hogy az eszköz valamilyen belsőleg összefüggő adathalmazt küldött. A kezelő úgy is dönthet, hogy az eseményeket az `input_sync` meghívásáig tárolja.

2. lista A register.c

```
#include <linux/input.h>
#include <linux/module.h>
#include <linux/init.h>

MODULE_LICENSE("GPL");

struct input_dev ex1_dev;

static int __init ex1_init(void)
{
    /* k l n sen  vatos indulÆs */
    memset(&ex1_dev, 0, sizeof(struct
           input_dev));
    init_input_dev(&ex1_dev);

    /* le r c mkØk beÆll tÆsa */
    ex1_dev.name = "Example 1 device";
    /* fut  rendszeren a phys egyedi*/
    ex1_dev.phys = "A/Fake/Path";
    ex1_dev.id.bustype = BUS_HOST;
    ex1_dev.id.vendor = 0x0001;
    ex1_dev.id.product = 0x0001;
    ex1_dev.id.version = 0x0100;

    /* ez az eszk z kØt kulccsal
       rendelkezik (A Øs B) */
    set_bit(EV_KEY, ex1_dev.evbit);
    set_bit(KEY_B, ex1_dev.keybit);
    set_bit(KEY_A, ex1_dev.keybit);

/* vØg l pedig bejegyezz k a bemeneti magba */
    input_register_device(&ex1_dev);

    return 0;
}

static void __exit ex1_exit(void)
{
    input_unregister_device(&ex1_dev);
}

module_init(ex1_init);
module_exit(ex1_exit);
```

Vessünk egy pillantást az utolsó eszközmeghajtó példánkra, ezúttal az után vizsgálódva, hogyan jönnek létre a viszonylagos adatok (*4. lista*). A példában egy olyan eszközmeghajtó látható, ami az egeret utánozza. A kezdeti beállítás szerint az eszköznek két viszonylagos tengelye van (REL_X és REL_Y) és egyetlen gombja (BTN_LEFT). Csakúgy, mint az előző példánkban, itt is egy időzítőt használunk, hogy meghívja az `ex3_timeout()` függvényt. Ez az időzítő ezt követően az `input_report_rel`-t hívja meg, hogy biztosítsa a viszonylagos mozgást (ötegységnyi lépésekkel – a viszonylagos elmozdulás a függvény harmadik értéke), ami harminc jobbra lépésből és harminc lefelé lépésből, valamint harminc balra lépésből és harminc felfelé lépésből áll, előidézve az egérmutató négyzetalakban történő mozgását. Hogy a mozgás érzését keltsük, az időtúllépést 20 milliszekundumra állítjuk. Jegyezzük meg, hogy az `input_sync`-et

3. lista Az aaaabbbb.c

```
struct input_dev ex2_dev;

void ex2_timeout(unsigned long
                 ➥unused/*UNUSED*/)
{
    int x;

    for (x=0;x<4;x++) {
      /* a vagy A betß */
      input_report_key(&ex2_dev, KEY_A, 1);
      input_sync(&ex2_dev);
      input_report_key(&ex2_dev, KEY_A, 0);
      input_sync(&ex2_dev);
    }
    for (x=0;x<4;x++) {
      /* b vagy B betß */
      input_report_key(&ex2_dev, KEY_B, 1);
      input_sync(&ex2_dev);
      input_report_key(&ex2_dev, KEY_B, 0);
      input_sync(&ex2_dev);
    }

  /* idız tı beÆll tÆsa nØgy mÆsodpercre */
    mod_timer(&ex2_dev.timer,jiffies+4*HZ );
}

static int __init ex2_init(void)
{

    ... elıkØsz tØs ...

    /* ismØtlıdı idız tı beÆll tÆsa */
    init_timer(&ex2_dev.timer);
    ex2_dev.timer.function = ex2_timeout;
    ex2_dev.timer.expires = jiffies + HZ;
    add_timer(&ex2_dev.timer);

    return 0;
}

static void __exit ex2_exit(void)
{
    del_timer_sync(&ex2_dev.timer);
    input_unregister_device(&ex2_dev);
}
```

mindig meg kell hívni, így biztosítva, hogy a bemeneti kezelők folyton összefüggő eseményeket kapjanak. Ez a követelmény az előző példánkban nem volt feltétlenül szükséges, de különösen fontos, ha olyan adatokat akarunk átadni a bemeneti magnak, mint például a viszonylagos mozgás, mert esetenként több tengelyre is szükség lehet a mozgás meghatározásához. Ha átlósan mozgatnád az egeret, ehhez hasonlót kellene megadnod:

```
...
input_report_rel(..., REL_X, ...);
input_report_rel(..., REL_Y, ...);
input_sync(...);
...
```

Ez biztosítja, hogy az egér átlósan mozogjon, ne pedig oldalra, majd fel.

Kezelők: a felhasználói szint

Ez előző szakaszban láthattuk, hogy az eszközmeghajtók az alkatrészek és a bemeneti mag között helyezkednek el. A részegység felől érkező eseményeket – ez többnyire a megszakításokat jelenti – bemeneti eseményekké alakítja át. A bemeneti események felhasználásához kezelőket használunk, amelyek a felhasználói szintű kapcsolódó felületet alkotják.
A bemeneti alrendszer tartalmazza mindazokat a kezelőket, amikre szükséged lehet: billentyűzetkezelőt a parancssor kezeléséhez, egérkezelőt az X Window System alatt futó alkalmazásokhoz, botkormánykezelőt a játékokhoz, valamint egy érintőképernyő-kezelőt is. Létezik egy általános célú kezelő is, az eseménykezelő, ami alapvetően a felhasználói területen futó programoknak a bemeneti eseményeit biztosítja. Ez azt jelenti, hogy szinte soha nincsen szükség rendszermagi kezelő írására, mivel ugyanezt a célt szolgálja az eseménykezelő és a hozzá tartozó felhasználói szintű kódok. Ezt a programozói felületet cikkünk második részében tárgyaljuk.

Köszönetnyilvánítás

Szeretném megköszönni ***Greg Kroah-Hartman*** és ***Vojtech Pavlik*** segítségét, amit a cikk első részének a megírásához nyújtottak.

A listák a 48. CD Magazin/USB könyvtárában találhatóak.

Linux Journal 2003. február, 107. szám

Brad Hards (bradh@frogmouth.net)
A Sigma Bravo technikai igazgatója egy szakértői szolgáltatásokat nyújtó kis cégnél, Canberrában. A Linux mellett repülőgéprendszerek összeillesztésével és minősítésével is foglalkoznak, csakúgy, mint GPS-szel és elektronikai hadieszközökkel.

KAPCSOLÓDÓ CÍMEK

A Linux bemeneti alrendszer elsősorban a 2.5-ös BitKeeper rendszermagban található meg. A BitKeeper sokoldalú rendszer, de ha csak a rendszermagot szeretnéd böngészni, a ➲ http://linus.bkbits.net:8080/linux-2.5 cím hasznos kiindulópontul szolgálhat.

Létezik egy kísérleti fejlesztési fa kizárólag a bemeneti alrendszerhez, amit a ➲ http://linux-input.bkbits.net:8080/linux-input címen érhetsz el.

Korábban a bemeneti alrendszert a ➲ http://linuxconsole.sourceforge.net címen lehetett elérni, de a rendszermaggal együtt átköltözött a BitKeeper alá. Bár az oldal nem tartalmaz túl sok leírást, található ott néhány hasznos folt a felhasználói szintű alkalmazásokhoz a CVS-fában.

Az AAAABBBB bővítmény egyik eredeti változata a ➲ http://www.wired.com/wired/5.08/linux_pr.html címen található meg.

A Red Hat Linux Kickstart program átalakítása

Hogyan hozzuk létre az igényeinknek megfelelő programokat tartalmazó CD-lemezt a gyors és könnyűszerrel elvégezhető telepítés számára?

A Linux-rendszer telepítése viszonylag könnyű feladat. Nemrég azonban az egymás után több gépre való telepítés feladata várt rám, ami időigényes tevékenység és magában hordozza a tévedések lehetőségét is. Ez a nehézség sújtotta a cégünket teljes egészében, sőt még azokat a cégeket is, amelyek a vállalkozásunkkal valamilyen függőségi kapcsolatban álltak. Emiatt kezdtem el használni a Red Hat Linux-változat Kickstart elnevezésű programját, vagyis hogy leegyszerűsítsem a telepítési folyamatot. Ez már önmagában is sokat segített, de még bőven volt mit finomítani. Amit tulajdonképpen szerettem volna, az nem más, mint egy önműködő telepítőcsomag, ami elfér egyetlen CD-lemezen: meghatározott részekre osztja fel a merevlemezt, és képes valamennyi frissített csomagot befogadni. Szerettem volna, ha el tudom indítani a telepítést, aztán magára hagyhatom a gépet, és csak akkor térek vissza hozzá, amikor már a telepítés befejeződött. Ennek a célnak az elérése érdekében a Kickstartot kiegészítettem a Red Hat telepítőprogramjának testre szabott változatával, az Anacondával.

Annak ellenére, hogy ez a program nem élvez hivatalos támogatást, a Red Hat több fejlesztőeszközt és leírást tett számomra hozzáférhetővé, hogy segítségemre legyen a telepítés testreszabásában. Ezek közül fogok bemutatni néhány módszert az alábbiakban, amelyek biztosítják az olvasó számára a kezdéshez szükséges információkat.

E cikkünkben az alábbi témákat fogjuk megtárgyalni: a telepített csomagok frissítése; a rendszertelepítés méretének csökkentése, hogy minden elférjen egyetlen CD-lemezen, és felhasználóiüzenet-képernyő létrehozása.

Általában véve az olvasónak tisztában kell lennie a Linux-telepítés elveivel. Továbbá azt is feltételezem, hogy semmiféle különleges vasat nem használunk a telepítés során, hiszen az további egyedi beállításokat igényelne.

A mintagép (build machine) telepítése

Az első lépés magának a mintagépnek az elkészítése. Minthogy a telepítőeszközök egy adott változatra jellemzők, mind a mintagépen, mind a célgép(ek)en ugyanannak a Red Hat Linux-változatnak kell működnie. Mostani példánkban a 8.0 változatszámmal ellátott Red Hat Linux-változatot fogjuk használni.

A korábbi Red Hat változatok és a 8.0-s rendszer között eltérések állnak fenn, ezeket tehát át kell vizsgálnod, ha valamelyik korábbi változatot használod.

A kívánt Linux-változatnak a mintagépre való telepítése után az Anaconda programcsomagjait kell telepíteni. Ezek az előzetes beállításoknak megfelelően nem lesznek telepítve, ezért kézi kiválasztás útján kell őket felhasználni. A szóban forgó csomagok a hivatalos Red Hat Linux-változat második lemezén találhatók meg, és az `anaconda`, `anaconda-runtime` valamint az `anaconda-help` nevet viselik – ennek az utóbbi csomagnak a telepítése nem kötelező.

A következő lépés annak a könyvtárszerkezetnek a létrehozása, ahová majd a telepítendő állományok fognak kerülni. A merevlemezrész méretét jól kell meghatározni, körülbelül 3 GB legyen. A merevlemezrész tényleges elhelyezése teljesen az egyéni ötletekre van bízva. A mostani példánkban a helye nem más, mint /RH80 . Ezen a könyvtáron belül minden egyes CD-lemez számára egy alkönyvtárat készítünk:

```
mkdir -p  /RH80/CD{1,2,3}
```

Most nem foglalkozunk a forrásprogramcsomagokkal, ezért a CD4 és CD5 lemezeket nyugodt szívvel kihagyhatjuk. Szükségünk lesz viszont a fentieken kívül egy további könyvtárra, amit az alábbi paranccsal hozzunk létre:

```
mkdir /RH80/ONE_CD
```

ide kerülnek a későbbiekben a kiválogatott programok. Ekkor az egyes CD-lemezek tartalmát a megfelelő névvel ellátott könyvtárba másoljuk. Csatlakoztassuk a rendszerhez az első CD-lemezt, utána pedig adjuk ki a következő parancsot:

```
cp -a /mnt/cdrom/* /RH80/CD1/
```

Ismételjük meg ezt a lépést a második (CD2), illetve a harmadik (CD3) lemez vonatkozásában. Most pedig másoljuk a CD-lemezeket jelző könyvtárak tartalmát a ONE_CD könyvtárba. Sokkal helyesebb azonban, ha csak közvetlen hivatkozást (hard link) hozunk létre, semmint hogy ténylegesen átmásoljuk az állományokat. Ilyen módon helyet takaríthatunk meg, ráadásul gyorsabban el lehet végezni ezt a műveletet.

```
cd /RH80
cp -al CD1/* ONE_CD/
cp -al CD{2,3}/RedHat/RPMS/* ONE_CD/RedHat/RPMS/
```

A képernyőn megjelenik egy kérdés, hogy a ***TRANS.TBL*** állomány felülírható-e. Nemmel is válaszolhatunk.

A csomagok kiválasztása

A ONE_CD könyvtár tartalmát addig nyirbáljuk, amíg fel nem fér egyetlen CD-lemezre – a korong kapacitását 700 MB-osnak feltételezem. Nem fogom részletekbe menően elmagyarázni, hogyan kell ezt a feladatot elvégezni, minthogy a Linux-terjesztésből eltávolítandó állományok listája telepítésről telepítésre változik. Hadd álljon itt mégis néhány tipp, hogyan lehet egy terjesztés méretét csökkenteni:

- A telepítendő rendszerbe nem fogadjuk be a programok forráskódját tartalmazó RPM-eket.
- Távolítsuk el a ***dosutils*** könyvtárat, minthogy ezek önműködő telepítések lesznek.
- A szükségtelen csomagok eltávolítása. Ez meglehetősen bonyolult feladat lehet, mivel biztosnak kell lennünk abban, hogy a függőségek továbbra is érintetlenek maradtak.

Fel kell jegyezni, hogy milyen állományokat távolítottunk el, hátha egyre-másra vissza kell lépni a korábbi változathoz. De akkor is szükséged lesz erre a listára, ha a későbbiekben a *comps.xml* állomány tartalmát kell szerkesztened.
A csomagok kiválasztásához a `base` és `core` csoportok összes csomagját a függőségeikkel együtt naplóállományba írtam, a *comps.xml* állományban leírtaknak megfelelően. Ennek az adatnak a beszerzéséhez a `getGroupPkgs.py` héjprogramot használtam.

```
cd /RH80/ONE_CD/RedHat/base
getGroupPkgs.py comps.xml > /RH80/pkglist
```

További csomagnevek fűzhetők hozzá az állomány végéhez. Amint elkészült az `syncRpms.py` héjprogram által készített listám, eltávolítottam a listán nem szereplő csomagokat.
A program paramétere a csomagokat tartalmazó könyvtár; a csomagok neveit tartalmazó listát a `getGroupPkgs.py` program hozza létre. Ez a héjprogram eltávolítja a listában nem szereplő csomagokat, és kiírja a csomagok neveit. Ezt a kimenetet egy állományba irányítjuk át, úgy, hogy a műveletről naplót kaphassunk.

```
cd /RH80
syncRpms.py ONE_CD/RedHat/RPMS/ pkglist
➥> pkgs_rem
```

A `du` parancs használata révén figyelemmel kísérhetjük a telepítendő rendszer méretét. A `-h` lehetőség földi halandók számára is érthető formában jeleníti meg az eredményt, a `-s` lehetőség viszont a teljes könyvtárfáról ad összegzést.

```
du -hs /RH80/ONE_CD
```

A *hdlist* állományok tényleges mérete újbóli létrehozásuk után csökken (lásd alább), mivel sok csomagot eltávolítottunk – ez viszont a CD méretét csökkenti.
A csomageltávolítás kényes része az, hogy megbonthatja a csomagfüggőségek rendszerét. Annak ellenére, hogy a `getGroupPkgs.py` a *comps.xml* állomány alapján feloldotta a függőségeket, nincs rá garancia, hogy az valóban helytálló. Ugyanakkor további csomagok beillesztése is megbonthatja a függőségeket. A függőségek helyességének ellenőrzésének egyik megbízható módja ideiglenes RPM-adatbázis létrehozása, majd az ez alapján kiválasztott csomagok próbatelepítésének végrehajtása.

```
cd /RH80/ONE_CD/RedHat/RPMS
mkdir /tmp/testdb
rpm --initdb --dbpath /tmp/testdb
rpm --test --dbpath /tmp/testdb -Uvh *.rpm
```

Keressük meg a kielégítetlen függőségeket jelző hibaüzeneteket. Ha ilyenek bukkannának fel, oldjuk fel a függőségi hibát csomagok hozzáadásával vagy akár azoknak az állományoknak az eltávolításával, amelyek a függőségi eltéréshez vezettek, s utána ismételjük meg a fenti próbatelepítést.
Ha a függőségi gondokat sikerült ilyen módon megoldani, máris hozzáláthatunk a rendszerünk frissítéséhez szükséges csomagok letöltéséhez. Helyezzük őket el egy önálló könyvtárba:

```
mkdir -p /RH80/updates/RPMS/
```

1. lista Kivonat a Kickstart-állományból

```
     .
     .
     .
# a lemezrıl mindegyik lemezrØsz let rlØse
clearpart --all --initlabel

# lemezfelosztÆsi informÆci
%include /tmp/partinfo
     .
     .
     .
%packages --resolvedeps
@Core
@Base

%pre
#!/bin/sh

#
# a lemezek felsorolÆsa a jellemzıkkel egy tt
#
set $(list-harddrives)
let numd=$#/2

# egyelıre annyit tudunk, mit kell
# elhelyezn nk kettınØl nem t bb meghajt n
d1=$1
d2=$3

if [ $numd == "2" ] ; then

echo " part /boot --fstype ext3 --size 75
➥--ondisk $d1
part / --fstype ext3 --size 1 --grow
➥--ondisk $d1 part swap --recommended
➥--ondisk $d1
part /home --fstype ext3 --size 1
➥-grow --ondisk $d2
" > /tmp/partinfo

else

echo "
part /boot --fstype ext3 --size 75
part swap --recommended
part / --fstype ext3 --size 2048 --grow
part /home --fstype ext3 --size 1024
" > /tmp/partinfo

fi
```

Távolítsuk el a régi állományokat a telepítési könyvtárból, a frissített állományokat pedig kapcsoljuk (link) a telepítési könyvtárhoz. Végezzük el ezt a műveletet minden egyes olyan frissített csomagra vonatkozóan, amelyben a *regi_rpm_allomany* az előző Linux-változathoz tartozó csomagot jelöli:

```
cd /RH80/ONE_CD/RedHat/RPMS/
rm regi_rpm_allomany.rpm
```

```
# ... az  sszes rØgi rpm eltÆvol tÆsa
cd /RH80/updates/RPMS/
cp -l uj_rpm_allomany.rpm
/RH80/ONE_CD/RedHat/RPMS/
# ... minden egyes .rpm-csomagra vonatkoz an
elvØgezni
```

Fel kell jegyezni, hogy mely csomagok lettek frissítve, hátha valamelyik frissítést vissza kell vonni. Jó ötlet a függőségek és méretek újbóli ellenőrzése, hátha megváltoztak, amióta frissítve lettek. Ezután ellenőrizzük minden csomag belső ellenőrzőösszegét (internal checksum) az RPM csomagkezelő `-K` kapcsolójával. Elsőként a kulcsot kell importálnunk:

```
cd /RH80/ONE_CD/RedHat/RPMS
rpm --import /usr/share/rhn/RPM-GPG-KEY
rpm -K *.rpm | grep "NOT *OK"
```

Ez az ellenőrzés nem szigorúan kötelező érvényű, ám minthogy csomagfrissítéseket töltöttünk le, ez ellenőrzi, hogy a csomagok érvényesek-e.

A telepítési állományok elkészítése

Ha már valamennyi állomány frissítve lett, újra létre kell hoznunk a ***hdlist*** állományokat. A csomagoknak csupán a fejlécét tartalmazzák ezek az állományok, amelyek az Anaconda számára lehetővé teszik, hogy gyorsabban jusson a fejlécadatokhoz. Minthogy csomagok frissítését végeztük el, ezeket az állományokat a `genhdlist` eszközzel ismét fel kell újítani; az eszköz az `anaconda-runtime` csomag része:

```
/usr/lib/anaconda-runtime/genhdlist \
/RH80/ONE_CD/
```

A ***comps.xml*** állomány kezeléséről sem szabad elfeledkeznünk. Ez az állomány csomagcsoportokat és csomagfüggőségeket határoz meg, bár erre az utóbbira nincs százszázalékos garancia. Az állományszerkezet éppen a 8.0-s Red Hat Linux változatban vált XML alapúvá, a korábbi változatokban csak rejtelmes címkéket tartalmazó közönséges szövegállomány volt fellelhető. Biztosítanunk kell, hogy a csoportokon belül meghatározott csomagok szerepeljenek a telepítésünkben. Ebben az esetben csak telepítendő csoportokkal kell foglalkoznunk. Abban az esetben, ha csomagok hiányoznak, vagy éppen ellenkezőleg, csomagok lettek a rendszerhez adva, a ***comps.xml*** nevű állományt kell szerkesztenünk.
Minthogy a `core` és `base` csoportnak valamennyi csomagját kiválasztottuk, nincs szükség ezen állomány szerkesztésére.
A Kickstart állomány `%packages` utasításában egyszerűen ezeket a csoportokat kell meghatároznunk. A Kickstart állományból készült kivonatot az *1. listában* vehetjük szemügyre. A `@Core` és `@Base` csoportokat a szó szoros értelmében kihagyhatjuk, hiszen ezek az előre megadott értékeknek megfelelően úgyis telepítve lesznek; itt kizárólag a példa kedvéért bukkantak fel.

Felhasználóiüzenet-képernyő létrehozása

Szeretnénk felhasználói üzenet képernyőt létrehozni, amin keresztül használati utasításokat adhatunk a felhasználó számára? Ezeket az üzeneteket a ***boot.img*** betöltési képállományban szokták tárolni, CD-ROM-on történő telepítésnél pedig pontosan az ***images*** könyvtárban. Ez DOS-állományrendszer formátumú, úgyhogy csatlakoztatni kell a rendszerhez, hogy hozzáférhessünk a tartalmához:

```
cd /RH80/ONE_CD/images
mount -o loop -t msdos boot.img /mnt/boot
```

Ha bekukkantunk a ***/mnt/boot*** könyvtárba, ott hat üzenetállományt fogunk találni: ***boot.msg, option.msg, general.msg, param.msg, rescue.msg, snake.msg***. Létrehozzuk a saját üzenetállományunkat, és a custom tetszőlegesen választott nevet adjuk neki. Minthogy a ***snake.msg*** állomány nincs használatban, a ***syslinux.cfg***-én belül ezt a bejegyzést cseréljük ki ***custom.msg***-re. Szerkesszük a ***syslinux.cfg*** állományt a ***/mnt/boot***-ban, és az F7 gombbal cseréljük ki a ***snake.msg***-ét F7 ***custom.msg***-re.
A ***syslinux.cfg*** állományon történt még néhány további módosítás, ezeket a 2. listában (48. CD Magazin/Kickstart könyvtár) tekinthetjük át. Az alapértelemezett bejegyzést `linux`-ról `ks`-re változtattuk. Ha időtúllépés fordul elő vagy a felhasználó a parancssornál leüti az Entert, akkor a `ks` címke lesz felhasználva. Ezenkívül az időtúllépés értékét 600-ról 60-ra csökkentettük, így a telepítés hamarabb megkezdődhet, amennyiben nem történik felhasználói adatbevitel.
A képernyőbejegyzés is megváltozott. A ***boot.msg*** bejelentkező képernyő – azt szerettük volna, ha a ***custom*** nevet viselő saját képernyőnk jelenik meg. A `ks` címke alatti `append` (hozzáfűzés) sort két pontosítással egészítettük ki. Az egyik a `text` kulcsszó, amivel lehetővé tesszük a szövegalapú telepítést. A másik pedig a `ks` kulcsszónak `ks=cdrom:/ks.cfg` -re történő megváltoztatása. Ez a kiegészítés pontosan kijelöli a Kickstart helyét, emiatt rendszerbetöltéskor a felhasználónak ezt szükségtelen megadnia a parancssorban. Ezután létrehozzuk a ***custom.msg*** üzenetállományunkat, amit a 3. listában (48 CD Magazin/Kickstart könyvtár) olvasgathatunk.
Az állomány tartalma színek használatával könnyebben áttekinthetővé tehető. Például az `^009Custom^002` jelsorozat a Custom színét kékre állítja be, a `^002` viszont visszaállítja a korábban használt betűszínt. A Kapcsolódó címeknél megtalálható syslinux referenciából részletesebben tájékozódhatunk.
Ha elkészültünk a custom üzenettel, már csak le kell választani a rendszerbetöltő képállományt (boot image).

```
umount /mnt/boot
```

A CD-lemez felépítése

Mielőtt a CD-lemezt megírnánk, hálózaton keresztül végzett telepítéssel talán ki szeretnénk próbálni az összeállítást: ennek mikéntjét a RedHat Linux Kickstart-leírásából ismerheted meg. Szeretnénk, ha az összeállítás képes lenne önműködően telepíteni magát, ezért a Kickstart-állományt magára a CD-lemezen is fel kell írnunk. A Kickstart-állományt bármilyen szövegszerkesztővel létre lehet hozni, de akár a Kickstart Configurator nevű grafikus felhasználói felületet is használhatjuk.
A `%pre` szakaszban egy héjprogrammal ellenőrizhetjük a merevlemezeket, és a meghajtók száma alapján dinamikusan létrehozhatjuk a lemezrészadatot. Azt a tényt használjuk ki, hogy a Kickstart a `%pre` szakaszt végrehajtja, utána újraolvassa a Kickstart-állományt. Amikor másodszor olvassa, akkor az értelmezésbe már befoglalja a ***/tmp/partinfo*** állományt is, ahol a `%include` utasítás található: lásd az 1. listát. A ***/tmp/partinfo*** a héjprogram kimenete. A `list-harddrives` parancsot használjuk, ami felsorolja a rendszer számára elérhető merevlemezeket azok méretével együtt. Amint a lemezrész létrejön, megszabadít bennünket attól, hogy többszörösen létre kellene hoznunk a Kickstart-állományokat, magukban hordozva a lemezrészjellemzőket.
A Kickstart-állományt létrejötte után nevezzük el ***ks.cfg***-nek, és helyezzük el a telepítőfánk gyökérkönyvtárában, vagyis a

A mkisofs által biztosított lehetőségek

-r Rock Ridge kiterjesztés, együtt a következőkkel: az állomány tulajdonosának valamilyen „használható” értékre való beállítása. Lehet, hogy meg kellene ismerned a -R lehetőséget, mivel -r-rel azonos, de a -R nem változtatja meg az állomány tulajdonosát.

-T A kapcsoló a TRANS.TBL állományt minden egyes könyvtárra vonatkozóan létrehozza. A használattól függően elképzelhető, hogy erre nincs is szükség.

-J A Joliet-kiterjesztések hasznosak, ha a CD-t más operációsrendszer-környezetben is használni szeretnénk.

-V A CD kötetcímkéje: ez az, ami akkor jelenik meg, ha a CD-lemezt a meghajtóba helyezzük.

-b Ez a kapcsoló jelzi, hogy a rendszerbetöltő képállományt kell használni, és a CD-lemezt „El Torito” típusú, rendszerbetöltésre alkalmas CD-ként kell elkészíteni.

-c Az „El Torito” típusú, rendszerbetöltésre alkalmas CD-lemeznél használatos indításkatalógus (boot catalog) elhelyezkedésének a forrásútvonalhoz viszonyított útvonala.

-o Az mkisofs parancs kimenetén megjelenő képállomány elhelyezkedése.

/RH80/ONE_CD/ könyvtárban. Egynél több Kickstart-állomány elkészítése és CD-lemezre írása is lehetséges. A különböző Kickstart-állományok szolgálhatják az „eltérő vasra” való telepítést. Most létrehozhatjuk a képállományt. Előző lépéseinknek köszönhetően a telepítendő rendszer elég kicsi lesz, és az összes frissített csomagot tartalmazza. A `mkisofs` program előállítja a képállományt, azután ezt felmásolhatjuk a CD-lemezre. A parancs, amivel az ISO képállományt előállíthatjuk, a következő:

```
cd /RH80/ONE_CD
mkisofs -r -T -J -V "My Custom Installation
➥CD" -b images/boot.img -c images/boot.cat
➥-o /RH80/mydist.iso /RH80/ONE_CD
```

Vess egy pillantást a használható lehetőségeket felsorakoztató táblázatra, mely a 48 CD Magazin/Kickstart könyvtárában található.

A `mkisofs` parancs utolsó jellemzője a lemeztartalmat hordozó forráskönyvtár, amit – például a saját telepítési könyvtárunkat – be kell foglalni a képállományba. Több más kapcsoló is használható, amit esetleg szeretnénk igénybe venni, például a `-A`, `-P` és a `-p` lehetőségek kiegészítő címkézési módot biztosítanak a képállomány számára. A `-m` és `-x` lehetőségek révén bizonyos könyvtárak vagy állománytípusok (file patterns) kizárhatók a képállományból – további tájékozódés végett olvasd el az mkisofs parancs súgóoldalát.

Ezután adjunk ellenőrző összeget (checksum) az ISO képállományhoz. Ez a művelet nem kötelező érvényű, de a végfelhasználók számára lehetőséget teremt, hogy ellenőrizzék CD-lemezük épségét. Az ISO képállományhoz ellenőrzőösszeget adó eszközt `implantisomd5`-nak hívják. Ezt a lépést a következő parancs kiadásával tehetjük meg:

```
implantisomd5 /RH80/mydist.iso
```

Egy társeszközzel az ISO képállományhoz adott ellenőrzőösszeget vizsgálhatjuk meg:

```
checkisomd5 /RH80/mydist.iso
```

A CD-lemezt telepítés közben is ellenőrizni lehet. A CD-ről való rendszerbetöltés után a felhasználó kiadhatja a parancsot:

```
linux mediacheck
```

Most már a képállomány felírható a CD-lemezre. Feltételezem, hogy a CD-író már telepítve van a rendszeredre. Az alábbiakban a `cdrecord` programot fogjuk használni, de ettől eltérő programokhoz is nyúlhatunk. A CD-íráshoz szükséges parancsot így kell kiadni:

```
cdrecord -v speed=4 dev=0,0,0 /RH80/mydist.iso
```

A `speed` (sebesség) és `dev` (készülék) lehetőségek csupán a rendszeredtől függnek. A `dev`-hez megadható készüléktípust a `cdrecord` parancs `-scanbus` ***paraméterével*** lehet meghatározni:

```
cdrecord -scanbus
```

A kész CD-lemez használata

Amint a képállomány CD-lemezre írása megtörtént, helyezd a lemezt a célgépbe, és indítsd el róla a rendszerbetöltést. A korábban készített és ***custom*** névre keresztelt saját üzenetet kell megkapnod. Ezen a ponton leütheted az ENTER billentyűt, vagy várakozhatsz, amíg időtúllépés nem lép fel a betöltési folyamatban. Ha ez bekövetkezik, akkor a betöltési folyamat az alapértelmezés szerinti címkét használja fel, vagyis azt, amelyet `ks`-ként (Kickstart) adtunk meg.

Ha mindent jól végeztünk, a telepítésnek felhasználói beavatkozás nélkül kell mennie. Tapasztalatom szerint a telepítés körülbelül tíz percig tart, de ez a vas jellemzőitől függően változhat.

Összefoglalás

A Kickstarttal és a saját céljainknak megfelelően átalakított Anacondával hatékony és rugalmas telepítés végezhető. Ez a telepítés nagymértékben javította a ciklusidőt és csökkentette a feladatkörömben (projekt) előforduló hibákat. Képes voltam több gépet több alkalommal telepíteni, szinte erőfeszítés nélkül. E cikkemben érintettem néhány módját annak, hogyan lehet kihasználni a Kickstartot és az Anacondát, ezeken kívül azonban még sok más lehetőség adott. Az érdeklődőket arra szeretném bátorítani, hogy tanulmányozzák át a Kapcsolódó címek között megadott leírást, és hogy bővebb információkért csatlakozzanak a Kickstart- és Anaconda-levelezőlistákhoz.

A cikkhez tartozó listák és címek megtalálhatóak a 48. CD Magazin/Kickstart könyvtárában.

Linux Journal 2003. április, 108. szám

Brett Schwarz
Seattle közelében él feleségével, fiával és a kutyájával. A számítógépes és vezeték nélküli rendszerek szaktanácsadója. Elérhető a saját honlapján, a ➲ http://www.bschwarz.com címen.

Felületi mintázatok a Blenderben (5. rész)

Ebben a részben megtanuljuk, hogyan lehet valódi tükröző felületet létrehozni a Blender segítségével.

Kezdeném talán a legegyszerűbb résszel, vagyis a képek és animációk mintázatként való felhasználásával. Állítsuk be a tárgy anyagát, kapcsoljunk át az F6 billentyűvel a mintázatok létrehozására alkalmas nézetbe. Itt a korábbiakban még nem ismertetett ***Image*** mintázatot kell választanunk, és az alábbi leírás alapján az igényeknek megfelelően kell majd beállítanunk a tulajdonságokat. A mintázat típusa alatt látható gombsorral kezdhetjük az ismerkedést.
Az `InterPol` kapcsolót arra használhatjuk, hogy amikor a tárgyra ráfeszítjük a mintázatot, a tárgy és a felhasznált kép mérete nem minden esetben azonos, tehát a képet a tárgy teljes befedéséhez a legtöbbször nagyítani kell. Ilyen esetekben a kép pontjait nem lenne célszerű csupán többszörözni – a szebb látvány eléréséhez két képpont közé színátmenetet képezhetünk. Ha például egy képet a kétszeresére nagyítunk, akkor egy piros és egy fehér képpont közé nem egyszerűen egy újabb piros pontot helyezünk el, hanem a két pont átlagát, vagyis egy rózsaszínű képpontot. Természetesen, ha több képpontot kell beilleszteni, akkor a következő képlet alapján számíthatjuk ki a megfelelő színt:

forrás_képpont+pillanatnyi_beillesztendő((cél_képpont-forrás_képpont) / beillesztendő_pontok_száma)*

Ezt nevezzük lineáris interpolációnak, ennek megvalósítására alkalmas az `InterPol` kapcsoló.
A következő kapcsoló a `UseAlpha`, ami arra jó, hogy a Blender az átlátszóságadatokat is tartalmazó képeken figyelembe vegye ezt a tulajdonságot. Ilyen képeket könnyedén előállíthatunk például a Gimppel. A képet TGA vagy PNG formátumban tárolva ez az adat megmarad és használható. Amennyiben alkalmazni kívánt képünk nem tartalmazza ezt az adatot, a Blender a `CalcAlpha` kapcsoló alkalmazásával képes ezeket az értékeket kiszámítani. Az átlátszóság kiszámításához a Blender a képpontok intenzitását használja fel, vagyis minél világosabb egy képpont, annál kevésbé lesz átlátszó a mintázat az előállításakor. A `NegAlpha` kapcsolóval a kiszámított vagy tárolt átlátszóságértékek inverzét használhatjuk, vagyis ebben az esetben a sötétebb képpontok lesznek jobban átlátszók, míg a világosabbak kevésbé. Természetesen a tárolt átlátszóságadatok esetében az érték nem függ a képpont világosságától, ott teljesen szabadon határozhatjuk meg a képpontok fényáteresztő képességét.
A `MipMap` kapcsolóval elérhetjük, hogy a Blender a képpontokat ne csupán két dimenzióban interpolálja, hanem a kamerától való távolság figyelembevételével több különböző felbontású képet készítsen a megadott mintázatból, és a kép kiszámítása során használja őket. Ennek az lehet az előnye, hogyha egy tárgy messzebb áll a kamerától, akkor nem kell akkora képpel számolni a megjelenítés során, vagyis gyorsabban kiszámítható a végeredmény. Viszont ahogy a tárgy közelebb kerül, a Blender a nagyobb felbontású mintát használja, tehát a tárgy mintázatának részletei is megjelennek. Ennek jó ellenpéldája a réges-régi Wolfenstein 3D program, ahol a falhoz közeledve a falminta nem részletesebbé, inkább élvezhetetlenebbé vált.
A `Rot90` kapcsolóval a mintázatként használt képet 90 fokkal forgathatjuk el, míg a `Movie` kapcsoló segítségével közölhetjük a Blenderrel, hogy nem egyszerű képről van szó, hanem egy animációról, ami képkockáról képkockára változik majd a tárgyon. Amikor animációt szeretnénk felhasználni, akkor a nézet közepén található `Frames` értékkel állíthatjuk be, hogy a kiszámított animációban mennyi képkockán keresztül kell ezt a mintázatot figyelembe venni, míg az alatta található `Offset` értékkel azt határozzuk meg, hogy azt a Blender a videórészlet hányadik képkockájától kezdve jelenítse meg a tárgy mintázataként.
A `Anti` kapcsolóval kapcsolhatjuk be az élek finomítását, amivel – hosszabb számítási időért cserébe – szebb eredményt kaphatunk.
A fentiekben tárgyalt kapcsolók alatt találjuk az egyik legfontosabb gombot ***Load Image*** felirattal, amivel megadhatjuk a Blendernek, hogy melyik képet vagy animációt szeretnénk a mintázat létrehozásához felhasználni. A gomb használatával egy állománylista jelenik meg, ahol kiválaszthatjuk a megfelelő képet, és a ***Load*** gombbal betölthetjük. Amennyiben olyan animációt adunk meg, amit a Blender nem képes betölteni, hibaüzenet figyelmeztet minket. Tapasztalataim szerint a Blenderrel készült animációk betölthetők, emellett a tömörítetlen képkockákat tartalmazó avi formátumú és a Motion JPEG tömörítéssel készült avi formátumú videórészletek használhatók.
A kép vagy animáció betöltése után a fájl elérési útja megjelenik a gomb melletti mezőben, és az állománynév mellett található ***Reload*** gomb használatával a Blender újra beolvassa az állományt. Ez a lehetőség akkor hasznos, amikor a pontos mintázat még nem készült el, vagyis folyamatos módosítás alatt áll, és szeretnénk megtekinteni a módosítások eredményét.
A képek betöltésére szolgáló gomb alatt találhatjuk a `Filter` értéket, amivel azt határozhatjuk meg, hogy az interpolált nagyítás és az előbbiekben tárgyalt `MipMap` szolgáltatás használatakor a Blender mekkora környezetet vegyen figyelembe a köztes értékek kiszámításakor.
A következő sorban lévő kapcsolókkal azt határozhatjuk meg, hogyha a kép vagy animáció kisebb, mint a tárgy, akkor a Blender hogyan számolja ki a tárgy fényáteresztő képességét. Az `Extend` kapcsolóval a kép szélén található színértékek a tárgyra is kiterjednek majd, a `Clip` hatására a mintázat által le nem fedett területek átlátszósága nulla lesz, a `ClipCube` használatával a kép köré rajzolható kockán kívüli területek átlátszósága lesz nulla értékű, míg a `Repeat` alkalmazásával a kép vagy animáció átlátszóságértékei is megismétlődnek a tárgyon. E kapcsolók alatti `XRepeat` és `YRepeat` értékekkel állíthatjuk be, hogy a mintázat a tárgyon a vízszintes és a függőleges tengely mentén hányszor ismétlődjön. A képek mintázatként való felhasználásakor még azt is fontos meghatároznunk, hogy a Blender mekkora területet vegyen figyelembe a tárgy mintázataként. Erre szolgál a `MinX`, `MinY`, `MaxX` és `MaxY` érték,

ezekkel pontosan beállíthatjuk a mintázat határait, és ezzel a megoldással elérhetjük azt is, hogy egy képen több tárgy mintázatát is megrajzolhassuk.
Nos, ennyi talán elegendő ahhoz, hogy tetszőleges képet helyezhessünk el egy tárgyon, meglehetősen pontosan és változatosan. Most ismerkedjünk meg azokkal a lehetőségekkel, amelyek segítenek a tükröző felületek előállításában. Azoknak, akik figyelemmel kísérték a sorozat korábbi részeit, talán feltűnt, hogy a mintázatok tárgyalásából kimaradt az `EnvMap` típusú felületi mintázat. Itt az alkalom, hogy ezt a hiányt pótoljam, és megismertessem olvasóimat ezzel a lehetőséggel

1. kép Egyszeres tükröződés

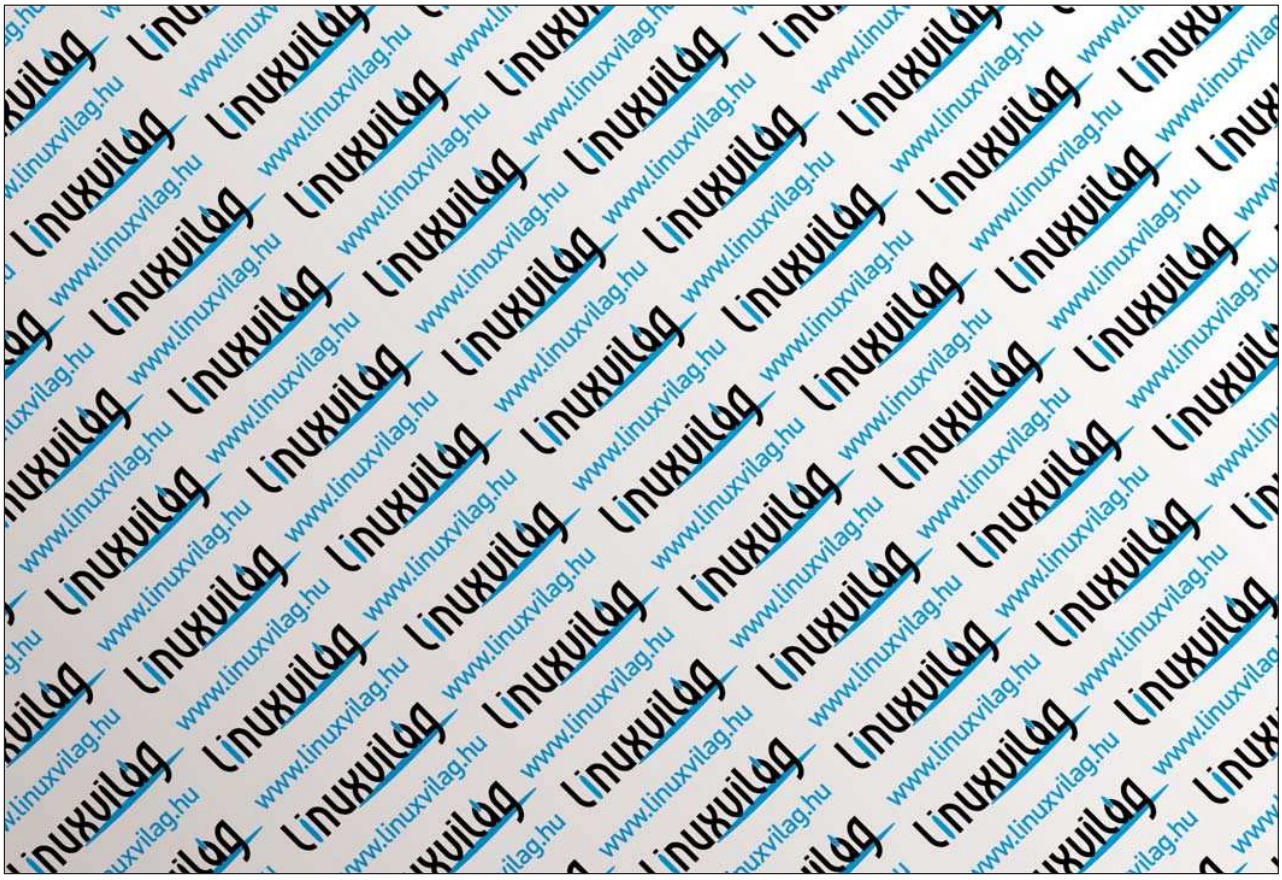
2. kép Többszörös tükröződés

és az eredményül kapott kép látványának nagyszerűségével. Az érthetőség kedvéért itt is az egyszerűbb látvánnyal kezdem. Amikor a Blendert elindítjuk, a jelenetben alapértelmezetten találunk egy tárgyat, használjuk fel ezt a tanulás során. Hamarosan ez a síkfelület lesz a tükröző felületünk. Hozzunk létre egy lámpát és egy gömböt, hogy legyen valami, ami tükröződik a síkon. A síklapot az ALT-M hatására megjelenő gombsor segítségével helyezzük át a második rétegre, majd mindkét réteget tegyük láthatóvá. Ezután jelöljük ki a síkot, és hozzunk létre egy új anyagot és egy új mintázatot is. A mintázat típusaként az `EnvMap` kapcsolót kell használnunk. Ez az a pont, amikor meg kell értenünk, hogy a Blender mi módon képezi le a környezetét visszatükröző tárgy mintázatát. Amikor egy ilyen mintázatot létrehozunk, a Blender egy virtuális kamerát helyez el a megadott ponton, és mindhárom tengellyel párhuzamosan két-két képet számít ki a jelenetről. Hármat pozitív irány felé néző kamerával, további hármat negatív irányba tekintő kamerával. A későbbiekben ezeket a képeket fogja használni mintázatként, mégpedig olyan módon, mintha egy kockába zártuk volna a tárgyat, aminek oldalain ezek a számított képek láthatóak. Az előbbiek alapján már látható, hogy a képek elkészítéséhez meg kell adni egy térbeli pontot, itt fogja a program elhelyezi a virtuális kamerát. Amikor a `EnvMap` kapcsolóval meghatároztuk a mintázat típusát, a nézetben megjelent egy `OB:` mező, amiben meg kell adni azt a tárgyat, ami ezt a pontot meghatározza. Azért, hogy e térbeli hely ne befolyásolja majd az összképet, a ***Főmenü/Add/Empty*** menüpontok kiválasztásával célszerű egy üres objektumot létrehozni. Ebből a pontból számítja ki tehát a program a képeket, ezért az előbbi `OB:` mezőbe írjuk be a látszólagos tárgy nevét (a tárgy neve „Empty”). A kamera elhelyezése után nyomjuk meg a SHIFT-N billentyűket, aminek a hatására megjelennek a kiválasztott tárgy tulajdonságai (ebben az esetben a kamera legyen a kiválasztott tárgy). Jegyezzük meg (írjuk fel) ezeket az értékeket, mert a helyes tükörkép előállításához a referenciaobjektumot pontosan a kamera alatt kell majd elhelyezni, csak a Z koordinátát kell ellentétes előjellel alkalmazni. Ez a megoldás akkor működőképes, ha a síklapot Z irányban nem mozgattuk el. Amennyiben elmozdítottuk, akkor a térbeli pont és a kamera közötti függőleges távolság felénél kell lennie, tehát a pont vagy a kamera magasságát ennek figyelembe vételével újra kell számolni. Feltételezve, hogy a síklapot nem mozdítottuk el, jelöljük ki a tükröződés számításához szükséges pontot, majd a SHIFT-N billentyűkombinációval állítsuk be a térbeli helyét. Az X és Y koordinátáknak meg kell egyezniük a kamera X és Y koordinátáival, a Z koordináta pedig legyen ugyanakkora, mint a kamera és a síklap távolsága, de azzal ellentétes előjelű (az értéke alapesetben megegyezik a kamera Z koordinátájával, az előjele azonban negatív). Ezek után már csak azt kell a program tudtára adni, hogy ezen a tárgyon valóban használni is szeretnénk a tükröződéshez kiszámított mintázatot, tehát jelöljük ki a síklapot, kapcsoljunk át az anyagszerkesztő nézetbe (F5 billentyű), és a nézet közepén kapcsoljuk be a `Refl` kapcsolót. Nézzük meg, mire jutottunk eddig: az F12-vel számíttassuk ki a képet. Láthatjuk, hogy a síklapon nem jelent meg a tükröződő gömb. Ez azért történhetett így, mert amikor a program a fentebb említett hat képet kiszámította, a tükröződés kiszámításához használt nézőpontból a síklap eltakarta a gombunkat. Természetesen erre is létezik megoldás. Kapcsoljunk vissza a mintázatokhoz, ahol a ***Don't render layer:*** felirat alatti gombsoron beállíthatjuk, hogy a program a számítások során milyen rétegeket hagyjon figyelmen kívül. Itt kapcsoljuk be a második réteget, hiszen a jelenet létrehozásakor a síklapot erre a rétegre helyeztük át. Ha most a végeredményt egyszerűen újból kiszámítanánk, az nem változna semmit, mert a szükséges képeket a program már az előbbi számolás során előállította. A korábban kiszámított adatokat töröljük a ***Free Data*** gomb használatával, és számoljuk ki újra a végeredményt. Ebben az esetben, ha mindent jól csináltunk, a síklapon meg kell jelennie a gömb tükörképének, ami azt jelenti, hogy továbbléphetünk a következő látvány megalkotásához.
Célunk most az lesz, hogy a síklapon két gömb tükörképét lehessen látni, s ezek közül az egyiken a másik gömb is visszatükröződjön. Hozzunk létre egy újabb gömböt a kamera látóterén belül, és a térbeli kurzor áthelyezése nélkül adjunk meg egy üres objektumot is. Az új gömböt helyezzük át a harmadik rétegre, és ezt is jelenítsük meg. A gömbnek adjunk anyagot, és mintázatként állítsuk be az `EnvMap` típust. Az anyagtulajdonságok között ne felejtsük el bekapcsolni a `Refl` kapcsolót. Tekintettel arra, hogy „Empty”nevű objektum már szerepel a jelenetben, ezt a nevet már nem adhatjuk meg kiindulási pontként a tükröződés számításához. Jelöljük ki az utoljára elhelyezett üres tárgyat (ez most a gömb belsejében található), és az F9 billentyűvel kapcsoljunk át a tárgyszerkesztő nézetbe. Itt a fejlécről leolvashatjuk a tárgy nevét (az eddigi lépéseket követve ez ***Empty.001*** lesz), amit az újabb gömb mintázatának kiindulási tárgyaként be kell írnunk az `OB:` mezőbe. Utolsó lépésként már csak azt kell megadnunk, hogy a Blender milyen rétegeket hagyjon figyelmen kívül a számítások során. Kapcsoljuk be a harmadik réteget, amin az új gömb található, hiszen az nem átlátszó – a tükröződés számításához használt képeken a gömb belseje jelenne meg. Ezek után a programmal számíttassuk ki a végeredményt.

SHIFT-N	Helyi koordináták és elfordulásértékek
ALT-M	Tárgy áthelyezése másik rétegre
F5	Anyagtulajdonságok
F6	Mintázatok
F9	Tárgyszerkesző nézet
F12	Kép kiszámítása

Láthatjuk, hogy ugyan a síklapon mindkét gömb tükörképe megjelent, a második gömbön szintén látható az első tükörképe, de mintha valami még nem lenne tökéletes. A képet jobban szemügyre véve feltűnik, hogy a második gömb tükörképén nem tükröződik az első gömb. Ennek az az oka, hogy a Blender a számítások során nem használ sugárkövetést, vagyis első lépésben nem számítja ki az egynél többszöri tükröződéseket. Ezt láthatjuk az *1. képen*. E nehézség megoldására „trükközödést"használunk. Miután ezt a kissé hibás képet kiszámítottuk, még egy lépést meg kell tennünk: töröljük a síklap kiszámított adatait – F6 → ***Free Data*** –, majd számítsuk őket újra. Így a második gömbről készülő képen már látható az első gömb képe, vagyis a kívánt végeredményt kapjuk meg, ezt láthatjuk a *2. képen*. További tükörtrükköket az ismertetett két módszer használatával készíthetünk. Ne feledjük el megadni a megfelelő kiindulási tárgyat, sem a megfelelő tárgyakon kétszer kiszámítani a környezetről készülő képeket, továbbá az anyagtulajdonságoknál bekapcsolni a `Refl` kapcsolót.
A gyakorlás során biztosan hamar feltűnik nekünk, hogy a tükörképek, bizony, meglehetősen alacsony felbontásban készülnek, azaz nem túl szépek. Természetesen a Blender erre is kínál megoldást, tekintsük csak meg a mintázatszerkesztő `CubeRes` értékét. Ennek alapbeállítása 100, ami nem túl nagy. Előfordulhat, hogy a tárgy–kamera-távolság olyan nagy, hogy a kis felbontású számítások is kielégítő eredményt adnak, de ha mégsem, akkor ezt az értéket szívünk és igényeink szerint legfeljebb 1000-re növelhetjük. Ezzel azt határozzuk meg, hogy a környezet leképezésekor használt kocka egy-egy oldalát alkotó kép milyen méretű legyen. Ez az érték képpontokban értendő. Amennyiben egy teljes animáció során szeretnénk a környezet leképezését alkalmazni (vagyis tükröződő felületeket számoltatni a programmal), a mintázat nézetében található `Static` kapcsolót kapcsoljuk át a mellette látható `Anim` elnevezésűre. Így a Blender minden képkocka kiszámítása előtt létrehozza majd a környezet mintázatát.
Végezetül a *táblázatban* láthatók a használt billentyűkombinációk, az újak és a már ismertek egy része egyaránt.
A CD-mellékleten (48. CD Magazin/Blender) található példákat tanulmányozva a Blender és a tükröződő felületek kapcsolata remélhetőleg könnyen érthető. Természetesen legjobb tudásom szerint szívesen válaszolok levélben feltett kérdésekre is. Legfontosabb tanácsom: továbbra se mulasszuk el folyamatos gyakorlást.

Fábián Zoltán (dzooli@freemail.hu, dzooli@yahoo.com) 27 éves, jelenleg programozóként dolgozik. Szabadidejében szívesen kirándul, túrázik. Emellett szeret rajzolni, érdekli a 3D grafika és a Linuxszal kapcsolatban minden olyan program és programnyelv, amit még nem ismer vagy nem próbált ki.



Irányítsunk mindent egy helyről a Synergyvel!

Kapcsoljuk a gépeket egymáshoz és önmagukhoz – KM kapcsolódobozok helyett – programmal!

Mit is jelent a Synergy? A szótár a következő meghatározást adja: „különálló elemek előnyös együttműködése”. A Synergy eszköz ezt az együttműködést olyan módon éri el, hogy két vagy több gép között TCP/IP-hálózaton keresztül képes egyetlen billentyűzetet és egeret átlátszó módon megosztani. Ezen felül a Synergy akár a kiválasztott szöveget és a vágólaptartalmat is képes megosztani az ICCCM (Inter-Client Communication Conventions Manual) elvárásainak megfelelően, Unicode-támogatással. A sorelemeléseket önműködően alakítja át a Unix és Windows formátumok közt, így a rendszerek közti vágás és másolás éppoly egyszerűvé válik, mintha egyetlen rendszeren dolgoznánk. Továbbá képes rávenni a képernyővédőket, hogy egyetértésben induljanak és álljanak le. Röviden minden számítógép a saját megjelenítőjét használja, nekünk csak az egeret kell a képernyő széléhez vinnünk, és máris átugrottunk a másikra. Az egész csaknem olyan érzés, mintha több számítógépen egyszerre egyetlen hatalmas munkafelületünk lenne.
A Synergy tulajdonképpen a billentyűzet, illetve az egér kapcsolódobozok programmal megvalósított megfelelője. Jelenleg Linuxon és Windowson fut, illetve előzetes változata a Solarist is támogatja. Ezek a felületek bármilyen összetételben használhatók. Ebből a cikkből azt tudhatjuk meg, hogyan telepíthetjük a Synergy rendszert két (vagy több) linuxos gépre. A beállítás általában csak néhány percet vesz igénybe.

Előkészítés és telepítés

Először is töltsük le a Synergy legutóbbi üzembiztos változatát a SourceForge oldalról (a *sourceforge.net/projects/synergy2* címről, vagy a 48-as CD Magazin/Synergy könytárából). Azután kövessük a megszokott módszert:

```
tar xzf synergy-X.Y.Z.tar.gz
cd synergy-X.Y.Z
./configure
make
su -c ·make install·
```

Itt az `X.Y.Z` a változatszámot jelenti. Az oldalon szintén elérhető, előre összeállított RPM-változatot is telepíthetjük, ha kívánjuk. A végrehajtható állományok a */usr/local/bin* könyvtárba kerülnek, hacsak más könyvtárat nem adunk meg a `configure`-nak. Ismételjük meg ezeket a lépéseket minden gépen, vagy a bináris állományokat (*synergyc* és *synergys*) egyszerűen másoljuk át.

A kiszolgáló beállítása

Válasszuk ki a kiszolgálót, azt a rendszert, amihez az egér és a billentyűzet fizikailag kapcsolódik. Ezen a rendszeren készítenünk kell egy Synergy-beállításfájlt, amiben megnevezzük a kiszolgálót, a kapcsolódásra jogosult számítógépeket (az ügyfeleket) és ezek virtuális képernyőkiosztását. Egyszerű szöveges állományról van szó, ami két kötelező és egy elhagyható szakaszból áll. Nézzünk egy példát a beállításfájlra:

```
section: screens
  guava:
  mango:
end
section: links
  guava:
    right = mango
    up    = guava
  mango:
    left  = guava
end
section: aliases
  guava:
    guava.tropical-fruit.org
  mango:
    mango.tropical-fruit.org
end
```

A *screens* (képernyő) szakasz egyszerűen felsorolja a kiszolgáló és az összes kapcsolódásra jogosult gép nevét. A *links* (kapcsolatok) szakasz a gépek képzeletbeli egymásmellettiségét adja meg. Például *guava* meghatározása szerint *mango* a jobb oldalán található, így amikor az egér kiszalad a *guava* képernyőjének jobb peremén, a *mango* képernyőjének ellenkező (jelen esetben a bal) oldalán jelenik meg újra. Minden számítógép legfeljebb egy-egy meghatározást tartalmazhat a következő kulcsszavak közül: `left` (bal), `right` (jobb), `up` (fel) és `down` (le). A számítógép saját magára is hivatkozhat; a fenti példában a *guava* felső részén kilépve az egér a *guava* alján jelenik meg újra. A befűzések nem lesznek önműködően szimmetrikusak. Ha a *mangóra* akarunk ugrani, kölcsönösen be kell állítanunk, hogy a *guava* a *mango* bal oldalán található. Ez a lehetőség akkor igazán hasznos, ha egyszerre több mint két gépet akarunk kezelni. Például legyen a harmadik gépünk neve *banan*, ami a *guava* és a *mango* felett helyezkedik el, de a *bananról* csak az egyikükre lehet lefele jönni.
Az *aliases* nevű harmadik szakasz elhagyható. Az ügyfelek csatlakozáskor megadják a gépnevüket (vagy a parancssorban megadott nevüket) a kiszolgálónak, így a kiszolgáló meg tudja őket keresni a beállításfájlban. Hálózati beállítástól függően némelyik rendszer teljes értékű tartománynevet ad vissza, mások viszont csak a gépnevüket. Az *alias* (álnév) szakaszban, mint a neve is sugallja, felsorolhatjuk, hogy milyen neveken ismerjük a számítógépeket. A fenti beállítás szerint a *mango* egyaránt kapcsolódhat *mango* vagy *mango.tropical-fruit.com* néven is. A kiszolgáló saját neve megállapításakor is végignézi az álneveket.
Bizonyára feltűnt, hogy a beállításfájl nem határozza meg, hogy a *guava* vagy a *mango* legyen-e a kiszolgáló. Erre ugyanis semmi szükség – ez a beállítás változtatás nélkül működik, bármelyik gép legyen is a kiszolgáló. Legyen mondjuk a *guava* gép a kiszolgálónk. Készítsük el saját beállításainkat a fenti példa alapján, majd mentsük a $*HOME*/*.synergy.conf* fájlba.

Az ügyfelek és kiszolgáló kipróbálása

Indítsuk el a Synergy kiszolgálót:

```
synergys -f -1
```

(Hamarosan a kapcsolók jelentését is megmagyarázzuk.) A kiszolgáló néhány üzenetet ír ki a héjprogram ablakába, és ha minden jól megy, készen várja a kapcsolatokat. A kiszolgálón belüli, önma-

gára mutató hivatkozások máris működnek. A *guava* gépen az egeret a képernyő tetejének tolva annak alulra kell ugrania. Minthogy a kiszolgálónk már fut, készen állunk az ügyfelek csatlakoztatására. Indítsuk el az ügyfelet a másik rendszerünkön (ebben a példánkban a *mango* gépen):

```
synergyc -f -1 --no-camp
guava
```

A *guava* nevet a saját kiszolgálónk nevével vagy hálózati címével kell helyettesítenünk. Az ügyfél szintén megjelenít néhány üzenetet, aztán vagy kapcsolódik a kiszolgálóra, vagy hibával kilép. Ha sikeresen csatlakozott, máris mindkét rendszerünkön használhatjuk az egeret, a billentyűzetet és a vágólapot. Ugyanígy próbáljuk csatlakoztatni a többi ügyfelet is.
Amennyiben a parancssori kapcsolók érvénytelenek vagy a beállításfájl hibás, a Synergy hibaüzenetet jelenít meg a héjprogramban, majd kilép. Ha a kiszolgáló- vagy az ügyfélkapcsolat valamilyen más okból kifolyólag sikertelen, akkor a hibát röviden ismertető ERROR vagy FATAL kezdetű naplóbejegyzést fogunk látni. Sajnos itt nincs hely arra, hogy valamennyi hibalehetőséget végignézzük, de az üzenet remélhetőleg elegendőnek fog bizonyulni, hogy megtaláljuk a hiba okát.
A fent használt parancssoros kapcsolók azt jelentették, hogy az ügyfél és a kiszolgáló az előtérben futhat és az üzeneteket a héjprogramban jelenítsék meg (`-f`), valamint – ha nem állandó hibajelenség bukkanna fel (`-1`) – a rendszer lépjen ki. Alapértelmezés szerint az ügyfél és a kiszolgáló is a háttérben fut, az üzenetek a `syslog`-ba íródnak, nem állandó hibák esetén pedig néhány másodperces várakozás után a rendszer újra próbálkozik. A `--no-camp` kapcsoló azt tudatja az ügyféllel, hogy lépjen ki, amennyiben a kiszolgáló az egyik kapcsolatot szándékosan lezárja. Egyébként az ügyfél egy kis tisztogatás után újra megpróbál kapcsolódni; erről az alábbiakban még ejtünk néhány szót. Létezik néhány további lehetőség is, a teljes lista megtekintéséhez adjuk ki a `--help` parancsot.

A Synergy önműködő indítása

Ha egyszer kipróbáltuk a kiszolgálót és az ügyfeleket, valószínűleg azt szeretnénk, ha a jövőben maguktól is elindulnának. A Synergynek szüksége van az X-kiszolgálóra, ezért ha az X felállása előtt indítjuk, nem fog működni. A legegyszerűbb módszer a Synergy önműködő elindítására, ha egy újabb sorral bővítjük a *$HOME/.xsession* állományunkat vagy az annak megfelelő X-folyamatot indító parancsfájlt. Általában az *.xsession* fájlból kapcsoló nélkül indíthatjuk el a Synergy-kiszolgálót, az ügyfelek egyetlen kapcsolója pedig a kiszolgáló neve lesz. A programok a háttérben fognak futni és maguk is kilépnek, amint az X-kiszolgáló kilép vagy újraindul.
Ezzel a beállítással csak az a gond, hogy a Synergy az `xdm` (vagy az ennek megfelelő `kdm` vagy `gdm`) által futtatott bejelentkező képernyőn sajnos nem fog futni. Ha rendelkezünk a szükséges jogosultságokkal, megjelenítéskezelőnket átállíthatjuk, hogy az X indításakor a Synergy rendszert is indítsa el. Először is a *$HOME/.synergy.conf* állományt másoljuk a */etc/synergy.conf* helyre (a második elején nincsen pont), hogy a megjelenítéskezelő megtalálhassa. Azután szerkesszük át az *Xsetup* parancsfájlt; a különféle terjesztések különféle helyeken tartják ezt az állományt, úgyhogy magunknak kell megkeresnünk. A parancsfájl vége felé, de még bármilyen kilépés- (exit) hívás elé szúrjunk be két sort. Két lehetőség között választhatunk. Azon a gépen, ahol az ügyfelet szeretnénk indítani, írjuk be a következő sorokat:

```
/usr/bin/killall synergyc
/usr/local/bin/synergyc guava
```

A guava szót helyettesítenünk kell a kiszolgálónk nevével. A kiszolgáló indításához csak ennyit kell beírnunk:

```
/usr/bin/killall synergys
/usr/local/bin/synergys
```

.Xsession fájlunkból ne felejtsük el eltávolítani a Synergyt indító valamennyi bejegyzést. Néhány megjelenítéskezelő biztonsági okokból lefoglalja a billentyűzetet, és a felhasználó bejelentkezéséig nem is ereszti el (ilyen az `xdm` és `kdm`, a `gdm` viszont nem). A Synergy kiszolgáló így bejelentkezés előtt nem képes megosztani az egeret és a billentyűzetet. Azt azonban nem akadályozza meg, hogy a Synergy ügyfél értelmezze az egér- és billentyűzetbemenetet; jelentkezzünk be a kiszolgálóra, majd a Synergy segítségével jelentkezzünk be az ügyfélre is.
A `--no-camp` kapcsoló nélkül indítva ügyfelünk 60 másodpercenként megpróbál kapcsolódni a kiszolgálóra, amíg sikerrel nem jár, úgyhogy az ügyfelet a kiszolgáló előtt is elindíthatjuk. Ezt a képességet a laptopgépeken ki is használhatjuk: folyamatosan futtassuk az ügyfelet. Amikor rákötjük az otthoni hálózatunkra, 60 másodpercen belül rákapcsolódik Synergy kiszolgálónkra. Ettől kezdve a kiszolgáló billentyűzetét és egerét használhatjuk a laptop eszközei helyett is.
Végül egy fontos megjegyzés a biztonsággal kapcsolatban. Az írás születésének pillanatában a Synergy még nincs semmilyen azonosító- vagy titkosítóképességgel felvértezve. Mivel az összes egér- és billentyűzetbementet továbbítja, beleértve a jelszavakat is, ne használjuk olyan hálózatokon, amikben nem bízunk meg. A Synergy jövőbeli változataiban hamarosan ezt a nehézséget is felszámolják.

Linux Journal 2003. április, 108. szám

Chris Schoeneman
(crs@groundhog.pair.com)
Grafikus programmérnökként dolgozik a Pixar Animációs Stúdióban. A Synergyn kívül ő alkotta a bzflaget is. California államban, Berkeleyben él.

Az rsync (2. rész)

Az rsync-modulok fájlrendszerszintű beállításai és kapcsolatok létrehozása.

Az előző alkalommal névtelen felhasználók számára helyeztünk üzembe egy rsync-kiszolgálót. Az első kódrészlet – a múlt hónapban is mintaként használt ***rsyncd.con***f állomány – néhány biztonsági szempontból hasznos beállítást szemléltet. Példánkhoz visszatérve most ejtsünk néhány szót az rsync-modulok (könyvtárak) fájlrendszerszintű beállításairól. A fő elvek ebben az esetben is megegyeznek a névtelen FTP `chroot`-környezetekben használtakkal. A különbség mindössze annyi, hogy a `chroot` használatához nem kell futtatható vagy beállítófájlokat bemásolni, mint azt néhány FTP-kiszolgálónál látni.

Az rsync beállítóállományában csak apróbb módosításokat kell végrehajtani az elérési útvonalak és engedélyezett állomások között, és máris a névtelen felhasználók rendelkezésére állhatunk. Ez még elég sovány dolog. Hogyan tudjuk fogadni a névtelen feltöltéseket, illetve miként hozhatunk létre külön modult a hitelesített felhasználók számára? A második kódrészlet mindkét kérdésre megadja a választ.

Először szükségünk lesz egy bejövő (a továbbiakban `bejovo`) nevű modulra, aminek az elérési útvonala */**home**/**bejovo***. Most is a nyilvánosan írható könyvtárak esetében (lásd „Tips for Securing Anonymous FTP” a „Building Secure Servers with Linux” című műben) alkalmazott elvek jutnak érvényre, egyetlen fontos különbséggel: a névtelen rsync szolgáltatás esetében a kérdéses könyvtárra mindenkinek futtatási és írási jogot kell adni, vagyis 0733-as módot kell beállítani. Ha ezt elmulasztjuk, a feltöltések anélkül lesznek sikertelenek, hogy a felhasználó erről bármiféle hibaüzenetet kapna, vagy a kiszolgáló naplójába bejegyzés kerülne.

Ebben az esetben is érdemes megfogadni néhány, az FTP-kiszolgálókra is érvényes tanácsot: a könyvtár tartalmának változását mindig kövessük figyelemmel, nehogy visszaéljenek a szolgáltatásunkkal, és a tartalma soha ne legyen bárki által olvasható. A feltöltött fájlokat a lehető leghamarabb – célszerűen a `cron`-ból – mozgassuk át egy nyilvánosan el nem érhető könyvtárba.

A `[bejovo]` csoportban az egyetlen új beállítás az átvitelek naplózása. Ezzel arra utasítható az rsync, hogy egy kicsit részletesebb naplót készítsen a fájlátviteli próbálkozásokról. Alapértelmezett esetben a beállítás értéke: `no` (nem). Emellett a már ismert `read-only` beállítás is `no` értéket kapott, felülbírálva ezzel az átfogó beállítások között szereplő `yes`-t (igen). Nincs hasonló eszköz arra, hogy az rsync tudomására hozzuk a könyvtár írható voltát. A lehetőségeket a könyvtárra megadott engedélyek szabják meg.

A példa második része egy korlátozott elérhetőségű, `Zenebolondok` nevű modult ad meg. Ebben három új beállítással ismerkedhetünk meg. Az első a `list`, ez adja meg, hogy a modul megjelenjen-e, amikor a távoli felhasználók a kiszolgálón elérhető modulok listáját kérdezik le. Alapértelmezett értéke `yes`.

A másik két beállítás, az `auth users` és a `secrets file` szabja meg a csatlakozó ügyfelek hitelesítésének módját. Az rsync hitelesítési összetevője, ami csak démonmódban érhető el, egy meglehetősen komoly, 128-bites, MD5 alapú kihívás-válaszadás jellegű eljárást használ. Ez két okból is jobb a normál FTP-hitelesítésnél. Egyrészt a rendszer nem továbbítja a jelszavakat a hálózaton keresztül, így nem lehet őket lehallgatni. Ettől még – elméletileg – nyers erővel (brute force), kivonatokkal kivitelezett támadást lehet indítani a kiszolgáló ellen!

Másrészt az rsync nem használja a rendszer hitelesítő adatait, a felhasználónév–jelszó párosokat saját állományban tárolja. Ezt az állományt kizárólag az rsync használja, semmilyen kapcsolatban nincs a */**etc**/**passwd*** vagy a */**etc**/**shadow*** állománnyal. Ha tehát valaki eredményesen támadja az rsyncet, a felhasználók rendszerazonosítói nem kerülnek közvetlen veszélybe, hacsak valaki nem végzetesen rosszul állította be az rsync könyvtárhozzáféréseit vagy az adott könyvtárakra vonatkozó jogosultságokat.

Maguk az adatátvitelek az FTP-hez hasonlóan titkosítás nélkül folynak – tehát igaz, hogy az rsync ellenőrzi a felhasználók adatait, ám az adatok épségét nem biztosítja, illetve bizalmasságukat sem védi a hallgatózóktól. Ilyen elvárások esetén SSH vagy Stunnel felett kell futtatni.

A `secrets file` beállítás adja meg az rsync felhasználónév-jelszó párosokat tároló állományának nevét. Ez hagyományosan a */**etc**/**rsyncd.secrets***, de gyakorlatilag bárhol, tetszőleges névvel elhelyezhető, és a ***.secrets*** végződés sem kötelező. Ennek a beállításnak nincs alapértelmezett értéke. Ha az `auth users` beállítást engedélyezed, akkor a `secrets file` értékét is meg kell adnod. Az alábbi részlet egy `secrets file` tartalmát szemlélteti:

```
watt:shyneePAT3
bell:d1ngplunkB00M!
```

A /etc/rsyncd.secrets mintaállomány tartalma

A második kódrészlet `auth users` beállítása adja meg, hogy a `secrets file`-ban szereplő felhasználók közül kik jogosultak a modul elérésére. Minden olyan ügyfélnek, aki ehhez a modulhoz próbál csatlakozni – feltéve, hogy átjutott a `hosts allow` és a `hosts deny` hozzáférés-vezérlési listák révén emelt akadályokon –, meg kell adnia nevét és jelszavát. Ne feledd pontosan megadni az érintett fájlokra és könyvtárakra vonatkozó jogosultságokat, ugyanis ezek szabják meg, hogy csatlakozás után a hitelesített felhasználók mit művelhetnek. Ha az `auth users` beállításnak nem adsz értéket, akkor a rendszer nem követeli meg a felhasználók hitelesítését, és a modul névtelen rsync-hozzáféréssel is elérhető lesz. Démonmódban ez az rsync alapértelmezett viselkedése.

Nagyjából ez az, amit egy névtelen és hitelesített felhasználók által egyaránt elérhető rsync-szolgáltatás beindításához tudnod kell. A parancssori és a beállítófájlban található további beállításokról – köztük az itt nem tárgyalt, a naplóüzenetek testreszabására használhatókról – az rsync(8) és az rsyncd.conf(5) súgóoldalak szolgáltatnak bővebb tájékoztatást.

1. lista Egy rsyncd.conf mintafájl

```
# kizÆr lag Ætfog  hat k rrel megadhat
# beÆll tÆsok
syslog facility = local5

# Ætfog  jellegß, de a modulokn Æl is
# megadhat  beÆll tÆsok
use chroot = yes
uid = nobody
gid = nobody
max connections = 20
timeout = 600
read only = yes

# pØldamodul:
[public]:
   path = /home/public_rsync
   comment = NyilvÆnos fÆjlok
   hosts allow = helyi.valami.org,
                 ➥10.18.3.12
   ignore nonreadable = yes
   refuse options = checksum
   dont compress = *
```

2. lista További rsyncd.conf modulok

```
[bejovo]
path = /home/bejovo
comment = Ide  rhatsz, de nem olvashatod
read only = no
ignore nonreadable = yes
transfer logging = yes

[Zenebolondok]
path = /home/cvs
comment = Zenebolondok CVS-tÆr
list = no
auth users = watt,bell
secrets file = /etc/rsyncd.secrets
```

Az rsync használata távoli rsync-kiszolgáló elérésére

Nehogy elfeledjem: még nem mondtam el, hogy rsync-kiszolgálóhoz hogyan kell ügyfélként kapcsolódni. Az írásmód roppant egyszerű, a távoli állomás nevének beírásakor egy helyett két kettőspontot kell használni, és a kívánt modulhoz abszolút helyett relatív elérési útvonalat kell megadni. Példaként vegyük elő az előző hónap felállását, amiben a helyi gépet helyinek, a távolit pedig távolinak hívtuk, és tegyük fel, hogy az ***ujcuccok.tgz*** fájlt szeretnénk letölteni, illetve a távoli gépen démonmódban fut az rsync. Emellett azt is feltesszük, hogy nem emlékszünk már rá, hogy a távoli gépen milyen nevű modul alatt találhatók az új állományok. Először tehát le kell kérdezni az elérhető modulok listáját:

```
[root@helyi darthelm ]# rsync tavoli::
public      NyilvÆnos fÆjlok
bejovo      Ide  rhatsz, de nem olvashatod
```

(Korántsem véletlen, hogy a modulok neve így alakul, a példákban is ezeket állítottuk be. Természetesen a Zenebolondok modul neve sem véletlenül nem jelenik meg a listában.)
A keresett könyvtár neve ***public***. Ezek után az alábbi parancscsal tudod az aktuális munkakönyvtáradba másolni az *ujcuccok.tg* fájlt:

```
[yodeldiva@helyi ~]# rsync
➥tavoli::public/ujcuccok.tgz
```

A kétszeres kettőspont és az elérési útvonal formátuma is eltér az SSH-módnál megismerttől. Míg az SSH abszolút elérési útvonalat vár a kettőspont mögött, az rsync démon egy modulnevet keres, ami a fájl elérési útjának gyökereként szolgál. Csak a példa kedvéért lássuk ugyanezt a parancsot SSH-módban:

```
[yodeldiva@helyi ~]# rsync -e ssh
➥tavoli:/home/public_rsync/ujcuccok.tgz
```

A két parancs természetesen nem teljesen egyenértékű, hiszen a távoli gép rsync démonfolyamatát a könyvtár tartalmának névtelen – vagyis hitelesítést nem végző – felhasználók számára való elérhetővé tételére állítottuk be, az SSH viszont minden alkalommal megköveteli a hitelesítést (igaz, ezt önműködővé is lehet tenni, ha nulla hosszúságú RSA vagy DSA kulcsot alkalmazunk; lásd a „Building Secure Servers with Linux” negyedik fejezetét). Nem is ez volt a lényeg, hanem az, hogy megmutassam az elérési útvonalak kezelésében jelentkező különbséget.

Az rsync bújtatása Stunnel segítségével

Az utolsó rsync használati mód, amiről szót ejtek, a démonmódban futó rsync és az Stunnel párosítása. Az Stunnel olyan általános célú TLS- vagy SSL-burkoló, ami bármelyik egyszerű TCP alapú átvitel titkosítással – és választható módon X.509 tanúsítvánnyal – védett beágyazására használható. Igaz, hogy SSH-módban futtatva az rsync titkosítással ruházható fel, ám elveszti démonmódban elérhető szolgáltatásait, amik közül a névtelen hozzáférés lehetőségét emelném ki. Az Stunnel segítségével legalább olyan jó titkosítás érhető el, mint SSH-val, de megtartható a névtelen kapcsolatok támogatása is.

Mi a helyzet az önhívással (recursio)?

Azzal kezdtem, hogy az rsync mennyire hasznos, ha nagyobb adatmennyiségeket – programarchívumokat, CVS-fákat – kell másolni, de összes példámban csak egy-egy állomány másolásáról volt szó. Ennek oka az, hogy elsősorban az rsync biztonságos használatát akartam bemutatni.
A rengeteg ügyféloldali (parancssori) rsync-kapcsoló felfedezésének örömét meghagytam neked, kedves olvasó. Mindenre kiterjedő leírást az rsync(8) súgóoldalon találsz. Különösen figyelemre méltó a `-a` (vagy `--archive`) kapcsoló, ami a `-rlptgoD` rövidítése, és a legtöbb fájltípusra – eszközökre és közvetett hivatkozásokra is – vonatkozóan önhívó működést ír elő; illetve a `-C` (vagy `--cvs-exclude`), ami az rsyncet CVS stílusú fájlkihagyási szabályok használatára utasítja a másolandó fájlok kiválasztásakor.

Az Stunnelről részletesen is olvashatsz a „Building Secure Servers with Linux” 5. fejezetében, ahol sok példában az rsyncet is viszontláthatod. Akadhatnak olyanok is, akik számára újdonság következik:

Ebben az esetben az alábbi lépéseket kell a kiszolgálóoldalon végrehajtani

1. Az ***rsyncd.conf*** beállításait a megszokott módon kell megadni.
2. Az rsyncet a `--port` kapcsolóval kell meghívni, és 873-tól eltérő kapuszámot kell megadni, például:
   `rsync -daemon --port=8730`
3. Egy Stunnel-figyelőt (listener) kell életre hívni a 873-as TCP-kapun, ami minden, ezen a kapun érkező bejövő kapcsolatot az előző lépésben megadott kapura fog irányítani.
4. Ha nem akarod, hogy bárki kedvére kapcsolódhasson a géphez, a ***hosts.allow*** állományban tiltsd le a 2. lépésben megadott kapura vonatkozó nem helyi kapcsolatokat. Emellett (vagy ehelyett) az IP Tables beállításait is hasonlóan módosíthatod.

Az ügyféloldalon az alábbiak szerint alakulnak a dolgok

1. Lépj be rendszergazdaként, állíts be egy Stunnel-figyelőt a 873-mas TCP-kapura (feltéve, hogy nincs rsync-kiszolgáló a helyi gépen, ami már használja ezt a kaput), ez az összes erre a kapura érkező bejövő kapcsolatot a távoli kiszolgáló 873-mas TCP-kapujára fogja irányítani.
2. Amikor kapcsolódni akarsz a távoli kiszolgálóhoz, a távoli kiszolgáló neveként `localhost`-ot kell megadnod. A helyi Stunnel-folyamat csatlakozik a kiszolgálóhoz, és rsync-csomagjaidat a távoli Stunnel-folyamat felé továbbítja. A távoli Stunnel-folyamat visszafejti az általad küldött rsync-csomagokat, és átadja őket a távoli rsync-démonnak. A válaszcsomagok természetesen ugyanezen a titkosított kapcsolaton keresztül érkeznek.

Mint látható, maga az rsync most is nagyon hasonló beállításokat használ, mint a névtelen csatlakozások esetében – a munka túlnyomó része az Stunnel-továbbítók beállításával kapcsolatos.

Linux Journal 2003. április, 108. szám

Mick Bauer (mick@visi.com)
Hálózati biztonsági tanácsadó az Upstream Solutions Inc.-nél Minneapolisban (Minnesota). Mick a szerzője a hamarosan megjelenő új O'Reilly könyvnek, amelynek címe „Building Secure With Linux”.

KAPCSOLÓDÓ CÍMEK

- http://www.rsync.org/
- http://samba.anu.edu.au/rsync
- http://sunsite.dk/info/guides/rsync/rsync-mirroring.html
- http://freshmeat.net/projects/rsync/

Héjprogramozás Linux alatt (2. rész)

Ismerkedjünk meg a héjváltozókkal, a parancssori kapcsolókkal és a feltételes utasításokkal!

Cikksorozatunk előző részében egy egyszerű héjprogramot kezdtünk el fejleszteni, ami a pillanatnyi könyvtár bejegyzései közül képes volt kiválogatni az alkönyvtárak neveit.
A forráskódja valahogy így festett:

```
1: #!/bin/sh
2: ls -l | grep ^d
3: echo " sszesen"‘ls -l | grep ^d | wc -l‘"
alk nyvtÆr."
```

A harmadik sorban látható parancsbehelyettesítés hajszálpontosan ugyanezt a műveletet tartalmazza, pusztán azért, mert a bejegyzéseket meg is akarjuk számolni. Nyilván az olvasó is érzi, hogy kétszer csinálni meg valamit ugyanabban a programban nem valami elegáns megoldás. Mennyivel szebb lenne, ha a második sor kimenetét átmenetileg tárolnánk valahol, és a harmadikban már csak fölhasználnánk! Nos, ezzel el is érkeztünk a változók használatához.
A héjprogramokban – akárcsak más programnyelvekben – gyakorlatilag tetszőleges számú változót használhatunk. Ezeket nem kell külön megadnunk, mivel abban a pillanatban, amikor értéket adunk nekik, önműködően létrejönnek. Előző példánk „elegánsabb kivitelben” a következőképpen fest:

```
1: #!/bin/sh
2: lista=‘ls -l | grep ^d‘
3: echo "$lista"
4: echo " sszesen"‘echo "$lista" | wc -l‘"
   alk nyvtÆr."
```

A második sor kimenete most egy `lista` nevű változóba kerül. Ezt a tartalmat a következő sorban egy `echo` paranccsal rögtön a képernyőre küldjük, a rákövetkezőben pedig ugyanezt a parancsot használva megszámoljuk és kiíratjuk, hogy a tartalma hány soros.
Figyeljük meg, hogy az új változót minden különösebb teketória nélkül vezettük be, egyszerűen a rá vonatkozó értékadással (2. sor). Amikor azonban a tartalmához akarunk hozzáférni, a $ dollárjelet kell közvetlenül a neve elé írni (3. és 4. sor).
Szintén érdemes megjegyezni, hogy a 4. sorban alkalmazott parancsbehelyettesítés szemmel láthatóan nem gátolja a változó tartalmára való hivatkozást. Ugyanakkor mindkét érték szerinti hivatkozást kettős idézőjelek közé zártuk – erre csak akkor van szükség, ha a változó tartalmában valamilyen különleges karakter (jelen esetben új sor) is található. A kettős idézőjel megakadályozza, hogy a héj értelmezze ezeket a karaktereket. Esetünkben, ha megszüntetjük az idézőjelezést, akkor az `echo` egyetlen hosszú sorként jeleníti meg a változó valójában többsoros tartalmát, és a `wc` is csak egy sort fog érzékelni (próbáljuk ki!).
Egy héjváltozóban bármit (karaktert, egy- vagy többsoros szöveget, számot) elhelyezhetünk, maga a héj azonban mindent szövegnek tekint. A héjváltozóknak tehát nincs a hagyományos értelemben vett típusuk. Értékadásnál a szóközöket is tartalmazó karakterláncot kettős idézőjelek közé kell zárni.
A fejlesztés következő lépéseként azt kellene megvalósítanunk, hogy programunk ne csak a pillanatnyi könyvtárban működjön, hanem parancssori kapcsolóként elfogadjon egy tetszőleges könyvtárnevet. Héjprogramokban a parancssori kapcsolókra a `$1`, `$2` stb. szimbólumokkal hivatkozhatunk; a parancssorban az egyes kapcsolók tartalmát egy vagy több szóköz választja el egymástól. Ha szóközt is tartalmazó karakterláncot akarunk egyetlen kapcsolóként megadni, akkor azt kettős idézőjelek közé kell zárnunk. Programunkban – legalábbis első megközelítésben – csupán a második sort kell módosítani:

```
2: lista=‘ls -l $1 | grep ^d‘
```

A dolog csaknem tökéletes, sőt új programunk még akkor is helyesen működik, ha elfelejtünk neki könyvtárnevet megadni. Ilyenkor „hagyományos módon” a pillanatnyi könyvtárral dolgozik. (Ebben ugyebár semmi természetfölötti nincs, hiszen a kapcsolók nélkül meghívott `ls` parancs is pontosan ezt teszi.) Egyéb, különleges helyzetekkel próbálkozva azonban hamar rájövünk, hogy akad még két apró gond. Ha a megadott könyvtárból egyáltalán nem nyílnak alkönyvtárak, programunk az összesítésnél akkor is azt állítja, hogy azok száma 1; ha pedig véletlenül nem létező könyvtárat adunk meg neki, akkor az `ls` parancs szabványos hibaüzenete jelenik meg, de a program az összesítést ilyenkor is elvégzi. Az utóbbi csak csúnya, az előbbi azonban kifejezetten hiba.
A téves összegzés okára kezdő héjprogramozók igen nehezen szoktak rájönni, és ez nem véletlen. Józanul gondolkodó ember ugyanis azt hinné, hogyha egy változóban nincs semmi, akkor az `echo` kimenetén is „semmi” fog megjelenni – ez azonban csak majdnem igaz. Az `echo` ugyanis alapértelmezésként akkor is kiküld egy újsor karaktert, ha semmi dolga nem lenne. Ezt pedig a wc boldogan megszámolja, hiszen számára az üres sor is sor. Az `echo` alapértelmezett viselkedését a `-n` kapcsolóval a 4. sorban lehet letiltani:

```
4: echo " sszesen"‘echo -n "$lista" | wc -l‘"
alk nyvtÆr."
```

A hibás kapcsolók kezeléshez természetesen egy logikai elágazást fogunk beiktatni, ami aszerint működik, hogy a parancssori kapcsolóként megadott néven létezik-e könyvtár. Ez azonban már sorozatunk következő részének a témája lesz.

Büki András (buki@vuk.chem.elte.hu)
Körülbelül kilenc éve dolgozik Linux operációs rendszerrel. Állandó szerzőtársa Prof. Dr. H. V. Kuksinak, akivel a Duna vagy a Tisza partján szoktak az élet és a tudomány viselt dolgairól töprengeni.

Operációs rendszerek (12. rész)

Írásunkban a Unix, illetve a Linux memóriakezelőnek a rendszerhívásokkal való kapcsolatát tárgyaljuk, valamint a jelek, a fork és az exec használatát mutatjuk be.

Ez idáig memóriakezeléssel foglalkoztunk, pontosabban bemutattuk, hogyan működik a virtuális memória. Szó volt ennek gyakorlati megvalósításáról is, méghozzá éles rendszereken bemutatva. A memóriakezelő feladata azonban még nem ér véget ezen a ponton – például kezelnie kell a rendszerhívásokat és a jelekkel is foglalkoznia kell. Folyamatstrukturált rendszerekben a memóriakezelőt az eszközközeli rendszermagvilág és a felhasználói réteg közötti határként is felfoghatjuk. (Ez részben a monolitikus rendszerekről is elmondható, de ott az effajta rétegződés nem ennyire nyilvánvaló). A memóriakezelő szorosan együttdolgozik a maggal, de nem foglalkozik az eszközök alacsony szintű használatával: a felhasználói folyamatokkal törődik.
Ezért a memóriakezelő életében nagyon fontos szerepet töltenek be a rendszerhívások. Emlékezzünk vissza sorozatunk első részére, ahol azt mondtuk, hogy a rendszerhívásokkal érhetjük el az operációs rendszer szolgáltatásait. Ezeket a felhasználótól érkező rendszerhívásokat valakinek értelmeznie kell, a kívánt feladatot végre kell hajtania (vagy egy alacsonyabb szinten lévő alrendszerrrel el kell végeztetnie), majd a végeredményt vissza kell juttatnia a rendszerhívást végrehajtó számára. Erre a feladatra valóban a memóriakezelő a legalkalmasabb.
A rendszerhívásokat egyébként két részre oszthatjuk fel: azokra, amiket a memóriakezelő szolgál ki, és azokra, amiket a fájlrendszer. Az előbbi csoportba nemcsak azok a szolgáltatások tartoznak, amelyek egy-egy memóriablokk lefoglalását, illetve felszabadítását teszik lehetővé, hanem olyanok is, mint például egy gyermekfolyamat létrehozása, egy alkalmazás elindítása vagy például egy jel elküldése. Sorozatunk mostani részében ezekkel a rendszerhívásokkal foglalkozunk, a fájlrendszerrel pedig a következő hónaptól kezdve ismerkedünk meg részletesebben is. Mindenekelőtt elevenítsünk fel pár fogalmat!

Néhány alapvető fogalom

A továbbiakban néhány olyan alapvető unixos fogalommal fogunk dobálózni, amelynek az egyik részéről már szóltunk, a másik részéről még nem – az a közös bennük, hogy nem árt tisztában lenni a jelentésükkel.
Az első ilyen fogalom a folyamatazonosító, vagyis a PID (Process ID). Egy olyan természetes számról van szó, amiből minden folyamat birtokol egyet, ugyanis a rendszer csak ennek alapján azonosítja magát a folyamatot. Talán felesleges megjegyeznünk, hogy két folyamatnak egyidejűleg nem lehet ugyanaz az azonosítója. Sőt, ha a rendszer egy azonosítót egyszer már kiosztott, azt soha többé nem osztja ki újból (pontosabban addig, ameddig újra nem indítjuk).
Nemcsak a felhasználók, de a folyamatok is csoportokba szervezhetők. Minden folyamat rendelkezik egy folyamat-csoport-azonosítóval is. Ilyen módon lehetővé válik, hogy nemcsak egy folyamatnak, hanem egy egész folyamatcsoportnak jeleket küldhessünk (lásd később).
A felhasználókat és a (felhasználói) csoportokat szintén számokkal azonosítjuk, ez az uid és a gid, velük valószínűleg már mindenki találkozott. Ami viszont magyarázatra szorulhat, az az, hogy mi is voltaképpen az az euid és az egid, azaz a tényleges (effektív) felhasználó-, illetve csoportazonosító. Ez az azonosító az, amelyik ténylegesen számít egy feladat elvégzésekor. Gondoljunk a `SETUID`-os programokra! Lényegtelen, hogy ki indítja el, a program akkor is megteheti mindazt, amit a tulajdonosa is megtehet. Ha tehát egy futtatható állomány a rendszergazda tulajdonában van, és rendelkezik a `SETUID` bittel, akkor a tényleges jogosultsága meg fog egyezni a rendszergazdai jogosultságokkal. (Ha a tényleges jogosultságok fogalma egy kicsit homályos, ne bánkódjunk, a következő részben a fájlrendszer kapcsán sokat fogunk vele foglalkozni.)
Az utolsó fogalom, amire szükség lesz, a tgrp-id, más néven terminál-csoportazonosító. Ennek az értéke annak a folyamatnak a PID-jével egyezik meg, amelyik az adott folyamathoz tartozó terminál eszközfájlját legelőször megnyitotta. (ez az esetek többségében nem más, mint a jelszóparancsfájl).
Most, hogy már mindent elsoroltunk, vágjunk is bele a közepébe!

A jelek

A jelekre már sokszor hivatkoztunk, arról azonban, hogy pontosan mik ezek és mire is valók, aligha beszéltünk részletesen. Most változtatunk a helyzeten, és elmesélünk mindent, amit a jelekről tudni érdemes.
Sorozatunk harmadik részében, amikor az IPC-vel (a folyamatok közötti kapcsolattartással) foglalkoztunk, több olyan nehézséget is bemutattunk, amelyeknek a feloldásához elengedhetetlen, hogy két folyamat egymással „beszélni” tudjon. Ezekben a példákban az volt a közös, hogy amikor egy folyamat már nem tudott tovább dolgozni (például az étkező filozófusok esetében nem állt rendelkezésre mind a két villa), akkor a folyamat blokkolt, és egészen addig várt, amíg valaki fel nem ébresztette.
Az előző mondat kulcsszava a „várt”. A gond ott van, hogy jó lenne, ha egy folyamattal akkor is tudnánk adatot közölni, amikor dolgozik, azaz éppen semmiféle bemenetre nem számít. Erre találták ki a jeleket – segítségükkel bármelyik időpillanatban jelezhetünk bizonyos dolgokat az éppen futó folyamatoknak, tekintet nélkül arra, hogy az éppen min munkálkodik. (Az már más kérdés, hogy az adott folyamat mit csinál a beérkező jelzéssel, az is előfordulhat, hogy figyelmen kívül hagyja.)
A jelek csak első hallásra tűnhetnek valamiféle titokzatos dolognak. Valójában teljesen hétköznapi dolgok a rendszer életében, sőt minden bizonnyal mindenki küldött már jelet egy folyamatnak. Gondoljunk csak arra, mit csinálunk, ha egy alkalmazás lefagyott. Vannak rendszerek, amikor a Reset gomb megnyomása az egyetlen kiút, de a Linux nem tartozik közéjük. Elegendő, ha egy `SIGTERM`, súlyosabb esetekben egy `SIGKILL` nevű jelet küldünk az adott folyamatnak, ami ennek hatására az fejvesztve menekül a memóriából. (A Unixban a jelek küldésére a `kill` utasítás szolgál. Sokan úgy gondolják, hogy ezt a parancsot egy folyamat megsemmisítésére találták ki, pedig eredeti feladata a jelküldés. Az más kérdés, hogy

szinte minden esetben a SIGTERM és a SIGKILL jel elküldésére használják, de bármi mást is továbbíthatunk a segítségével. A jelekről listát a `kill -l` utasítással kaphatunk.)
Hogy egy alkalmazás mit tesz a beérkező jelekkel, az kizárólag a programozón múlik. Mindenesetre a jelek kezelése meglehetősen kényelmes feladat: csupán annyi a feladatunk, hogy a program indulásakor a SIGACTION nevű rendszerhívás segítségével átadjuk annak az eljárásnak a címét, amelyik majd a jel kezeléséért felelős lesz. Például a SIGTERM jelhez is rendelhetünk egy ilyen eljárást, ami nem csinál mást, mint lezárja az összes megnyitott állományt, ezután visszaadja a folyamat által foglalt memóriát. Ennek köszönhetően, ha véletlenül végtelen ciklusba kerül a folyamat, a SIGTERM jel hatására ki fog lépni.
Ugyanennek a rendszerhívásnak a segítségével lehetőségünk nyílik a beérkező jelek figyelmen kívül hagyására is. Ilyenkor hiába küldözgetünk lázasan jeleket, semmi sem fog történni.
A jeleket blokkolhatjuk is, ez a folyamat annyiban különbözik az előzőtől, hogy a beérkező jelek tárolva lesznek, és a folyamat csak a blokkolás megszüntetése után kapja meg őket.
Ha nem készítünk ilyen eljárást, akkor az adott folyamat figyelmem kívül fogja hagyni a beérkező jeleket, még a SIGTERM-et is. (Pontosabban, ha egy folyamat a SIGACTION rendszerhívás segítségével nem rendel jelkezelő eljárást egy jelhez, akkor az adott jel beérkezésekor a memóriakezelő egy alapértelmezett tevékenységet fog elvégezni. Hogy ez pontosan mi is, az az adott jeltől függ, de általában két dolgot jelenthet: vagy nem történik semmi, vagy kilövi a folyamatot.)
Létezik azonban egy jel, amit egyetlen folyamat sem vehet semmibe: ez pedig a SIGKILL. Ezt a jelet ugyanis nem a folyamatnak kell kezelnie, hanem a memóriakezelőnek. A folyamat nem is tud róla, hogy neki SIGKILL-t küldtek, és esélye sincs védekezni ellene. Ilyen jelre nyilvánvalóan szükség van, mert enélkül lehetetlen lenne megszabadulni az olyan lefagyott alkalmazásoktól, amik nem válaszolnak a SIGTERM-re. Viszont érdemes a beragadt programok ellen bevethető legeslegutolsó „fegyvernek” meghagyni, ugyanis ilyenkor nincs lehetősége a memóriában lévő adatokat a lemezre írni, míg a SIGTERM esetében ezt még megteheti.

A jelek és a memóriakezelő kapcsolata

Most már tudjuk, mi történik akkor, ha egy folyamathoz jel érkezik. Az azonban még mindig homályos folt számunkra, hogy miként juttatja őket a rendszer célba.
A jelek alapvetően kétféleképpen keletkezhetnek: vagy egy folyamat által, a `kill` rendszerhívás segítségével (ez az eset megegyezik azzal, amikor a felhasználó kiadja a `kill` utasítást), vagy maga a rendszermag hívja életre őket. A rendszer számára lényegtelen, miképpen is keletkezett az adott jel, a célbajuttatás módja minden esetben megegyezik.
Először is meg kell állapítanunk, hogy ki kinek küldte az adott jelet, és van-e hozzá jogosultsága. A jogosultság ellenőrzésekor az az alapelv, hogy egy rendszergazdai folyamat mindenkinek küldhet jelet, viszont mindenki más csak azoknak, akiknek a felhasználói azonosítói megegyeznek az sajátjáéval. Például minden általunk elindított alkalmazásnak küldhetünk SIGKILL-t, de más felhasználó folyamatait ilyen módon nem tudjuk kiiktatni. A rendszergazdára ilyen megkötés nem vonatkozik, ő bárkinek a folyamatát leállíthatja. A dolog azért nem ilyen egyszerű: a memóriakezelőnek olyasmire is figyelnie kell, hogy a címzett véletlenül nem zombifolyamat-e (mert ilyennek nincs értelme jelet küldeni).
Ezután ellenőrizni kell, hogy a címzett folyamat az adott jelet figyelembe veszi-e, illetve érvényben van-e a jelek blokkolása. Ha érvényben van, akkor fel kell jegyezni, hogy milyen jelet kapott, és – amint a blokkolás megszűnik – ismételten el kell küldenie a memóriakezelőnek (természetesen a SIGTERM jelre a blokkolás nem vonatkozik).
Ha a blokkolás érvénytelen, akkor két eset lehetséges: a folyamat vagy foglalkozik az adott jellel, vagy figyelmen kívül hagyja. Először nézzük meg az előbbi esetet! Ilyenkor a

memóriakezelőnek ki kell számítania, hogy a címzett folyamat pontosan hol is helyezkedik el a memóriában, továbbá meg kell hívnia a jelkezelő eljárást. Ez azonban nem egyszerű művelet, ugyanis előbb mentenie kell a folyamat pillanatnyi állapotát (erre azért van szükség, hogy a folyamat ugyanonnan folytathassa munkáját, ahol a jel beérkezése előtt tartott). Erre a folyamat vermét szokás használni. Ezzel az a gond, hogy korántsem biztos, hogy a veremben elegendő szabad hely van az összes adat mentésére, ezért még mielőtt bármi érdemlegeset cselekednénk, ellenőrizni kell, hogy valóban elfér-e minden a veremben. Ha nincs elég szabad hely, akkor baj van, mert a rendszer nem tehet mást, minthogy kilövi a folyamatot (szerencsére ma már elég ritkán fordul elő, hogy a folyamat verme megtelik). A folyamat állapotának mentését a memóriakezelő nem tudja elvégezni, ezért az alacsonyabb szinten lévő rendszermaghoz fordul. A jelkezelő eljárás csak ezután kerül meghívásra. Miután a folyamat a szükséges teendőket elvégezte, meghívja a SIGRETURN rendszerhívást, aminek hatására a veremből visszaállítódik a régi állapot (ezt is a mag végzi).
Ha viszont a folyamat nem rendelkezik jelkezelő eljárással, akkor a memóriakezelő az adott jelhez tartozó „alapértelmezett tevékenységet” fogja elvégezni. Ez két dolgot jelenthet: vagy nem fog történni semmi (azaz a memóriakezelő figyelmen kívül hagyja a jelet), vagy kilövi a folyamatot. Az utóbbi esetben mindig készül egy úgynevezett core dump állomány, ami az adott folyamat memóriatérképét tartalmazza (ehhez a fájlrendszer közreműködése is szükséges).

Amikor egy folyamat elindul…

A Unixban kétféleképpen keletkezhet új folyamat. Létrejöhet úgy, hogy egy – már létező – folyamat szétágazik, azaz a `fork` rendszerhívás segítségével gyermekfolyamatot hoz létre. De megszülethet olyan módon is, hogy egy folyamat egy másik programot indít el maga helyett. Erre az `exec` nevű rendszerhívás alkalmas. A két rendszerhívás tehát teljesen másra használható és másképpen is működik, azonban van bennük két közös dolog: mindkettő új folyamatot

1. lista Példa egy gyermekfolyamat létrehozására

```
result = fork(); // gyermekfolyamat
lØtrehozÆsa
if (result &lt; 0)  // ha a visszatØrØsi
                   // ØrtØk negat v ...
{
  // hiba zenetet
} else {
  if (result == 0)
  {
     // ezt fogja csinÆlni a
     // gyermekfolyamat
  }
else
  {
    // ezt pedig a sz lı
}
```

hoz létre és memóriafoglalást hajt végre. A továbbiakban mind a kettővel megismerkedünk.

Gyermekfolyamatok

Egy folyamatnak tehát a `FORK` rendszerhívás segítségével gyermekfolyamatok létrehozására nyílik lehetősége. A gyermekfolyamat csaknem pontos másolata lesz a szülőfolyamatnak: ugyanannyi memóriát foglal el és ugyanazt a kódot is használja. A gyermek azonban egy különálló folyamat, azaz a szülővel (és a többi folyamattal) párhuzamosan fog futni.
Hogy megértsük, mit is jelent ez a gyakorlatban, vessük egy pillantást az *1. listára*, ahol egy gyermekfolyamat létrehozására láthatunk példát. Azok kedvéért, akik nem jártasak a C programozási nyelvben, röviden elmeséljük, mi történik. Először is meghívjuk a `fork` rendszerhívást (pontosabban a `fork()` könyvtári eljárást, ami majd intézkedik, hogy a `fork` rendszerhívás végrehajtódjon). A művelet végeredménye egy szám lesz (amit a `result` nevű változóban tárolunk). Ha az értéke negatív, akkor nem jártunk sikerrel, például azért, mert nem volt elég szabad memória. Ilyenkor nincs más teendőnk, mint egy hibaüzenetet kiíratni és kilépni. Ha azonban a `result` értéke 0 vagy nagyobb, akkor a gyermekfolyamat megszületett. Ne felejtsük el, hogy a gyermek és a szülő kódja ugyanaz, tehát innentől kezdve a gyermek is ezt a kódot hajtja végre! A különbség csupán annyi, hogy a gyermek esetében a változó értéke 0, míg a szülőnél a létrejött gyermek folyamatazonosítóját (PID-jét) tartalmazza. Ilyen módon lehetőség nyílik annak eldöntésére, hogy az adott kódot most a szülő- vagy a gyermekfolyamat hajtja-e végre.
Meg kell jegyeznünk, hogy a `fork` rendszerhívást „gonosz” dolgokra is fel lehet használni, például arra, hogy egy folyamat nagyon sok példányban előállíthassa magát, leterhelve ezáltal a processzort és „felzabálva” a szabad memóriát. Az ilyen programokat forkbombáknak nevezzük. Hatékonyan védekezni ellenük csak az erőforrások használatának korlátozásával lehet, de a jelenlegi Linux-terjesztések szerencsére már alapból védettek az efféle támadások ellen.
A gyermekfolyamat nemcsak a szülő kódját, hanem a jogosultságait is örökli. Ha nagyon pontosak szeretnénk lenni, akkor ezt valahogy úgy lehetne megfogalmazni, hogy a gyermekfolyamatok öröklik a szülőt futtató felhasználó- és csoportazonosítót, továbbá a tényleges felhasználó- és csoportazonosítókat is.

2. lista
Példa az exec rendszerhívás végrehajtására az execle() könyvtári függvény segítségével. Ez a sor a /bin/ls program meghívását eredményezi – ugyanaz, mintha a parancsfájlban kiadtuk volna az „ls -l claudia.jpg” utasítást.

```
r=execle( /bin/ls","ls","-l",
➥"claudia.jpg",(char *)0, envp);

Az elsı ØrtØk a binÆris ÆllomÆny nevØt
tartalmazza. A k vetkezı ØrtØk magÆnak a
parancsnak a neve (ez egyØbkØnt elhagyhat ),
a harmadik Øs a negyedik ØrtØkkel a
vØgrehajtand  parancs kapcsol it adhatjuk
meg. Mivel t bb ØrtØket is meg lehet adni,
sz ksØg van rÆ, hogy jelezni tudjuk, mikor
Ør vØget a felsorolÆs. Erre szolgÆl a
(char *)0. Az utols  ØrtØk az œj programnak
Ætadand  parancsfÆjlvÆltoz kra mutat.
```

3. lista Példa az exec és a fork rendszerhívások ötvözésére

```
int result;
if (fork() == 0)
  { execle("/bin/ls","ls","-l",(char
*)0,envp); }
 else {
    wait(&result); }
```

A jogosultságokon kívül a munkakönyvtárban és a jelkezelő eljárásokban sincs különbség a gyermek és a szülő között.
A gyermek ugyanakkor az ütemező számára független folyamat, így egyedi folyamatazonosítóval (PID-del) kell rendelkeznie. Akad még pár dolog, amiből a gyermekfolyamat sajáttal bír, ilyenek például a fájlleírók, de erről majd a fájlrendszer tárgyalásakor beszélünk részletesen.

Az exec rendszerhívás

A `fork` rendszerhívást abban az esetben használhatjuk, ha a gyermeknek valamilyen a szülőfolyamathoz hasonló feladatot kell ellátnia. A gyakorlatban azonban ilyesmi viszonylag ritkán fordul elő. A helyzet az, hogy legtöbbször valamiféle a szülőfolyamattól teljesen független dolgot kell tennie.
Sőt van olyan eset is, amikor csak a folyamat futása közben dől el, hogy pontosan mi is lesz a feladata. (Gondoljunk a parancsértelmezőre, amiben egy – a felhasználó által megadott – állományban lévő kódot kell végrehajtatni.) Ilyenkor mindig az `exec` rendszerhívást kell alkalmazni, ami mind közül a legbonyolultabb.
(A `fork` és az `exec` közötti különbséget a legkönnyebben úgy érthetjük meg, hogy a `fork`-ot úgy képzeljük el, mint a saját programunk szétágaztatását, azaz a különböző részfeladatokat egymással párhuzamosan, külön folyamatként végeztetjük el. Az `exec` esetében viszont egy, a miénktől teljesen független programot hívunk meg. Fontos, hogy az ilyenkor nem a mi folyamatunkkal párhuzamosan, hanem ahelyett fut. A mi programunk csak akkor folytatódik, ha az `exec`-kel indított alkalmazás már lefutott.)
Az `exec` rendszerhívásnak nemcsak a megvalósítása nehéz,

hanem a meghívása is egy kissé bonyodalmas. Ráadásul nemcsak egy, hanem többféle könyvtári eljárás is létezik az `exec` rendszerhívás végrehajtására. Leggyakrabban az `execle()`-t szokás használni, aminek használatára – érdekesség gyanánt – a *2. listán* láthatunk példát.
Az esetek többségében a `fork`-ot és az `exec` rendszerhívást a *3. listán* látható módon egyszerre szokás használni. Az elv nagyon egyszerű: a `fork`-kal egy gyermekfolyamatot hozunk létre, ami nem tesz mást, minthogy az `exec`-kel elindítja a megadott programot. Az ilyen módon indított alkalmazás a szülőfolyamattal párhuzamosan képes futni. Ez az alapja a Unix-parancsértelmezőknek (például sh, bash) is.
Felmerülhet a kérdés, hogy miért nincs olyan rendszerhívás, ami ilyen formában egyesíti a `fork` és az `exec` tulajdonságait. A válasz az, hogy felesleges lenne. Ezzel csak bonyolítanánk az életünket, mivel gondoskodnunk kellene a szabványos ki- és bemenet (az az eszköz, ahová a program az eredményt kiírja, illetve ahonnan az adatokat bekéri) oda-visszairányításáról. Ehelyett a létrejött gyermekfolyamatnak csak le kell zárnia a pillanatnyi be- és kimenetet, majd egy új megnyitása után meghívni az `exec`-et (a gyermekfolyamatok ugyanis az átirányított ki- és bemenetet is öröklik).
Az `exec` rendszerhívás megvalósítása jóval bonyolultabb a `fork`-énál. Az utóbbinál „mindössze” annyi a feladat, hogy a folyamattáblában létre kell hozni egy új bejegyzést, majd a szülő adatait be kell másolni a gyermekhez. Az `exec` esetében azonban már egy teljesen új memóriatérképet kell létrehozni, továbbá a fájlrendszerrel is együtt kell működni, mivel mégis-csak a lemezről olvassuk be a futtatni kívánt kódot.
Amikor elindítunk egy programot (például a parancssorban kiadjuk az `ls` utasítást, ami a */bin/ls* program meghívását eredményezi), akkor mind a memóriakezelőnek, mind a fájlrendszernek rögtön ellenőrzést kell végrehajtania. Először is meg kell állapítaniuk, hogy egyáltalán van-e jogosultságunk futtatni az adott állományt (ez a fájlrendszer feladata), továbbá utána kell járniuk, hogy van-e elég szabad memória (ezt pedig a memóriakezelő intézi). Ha ezek közül egy is negatív eredménnyel jár, a program indítása meghiúsul.
Azt, hogy pontosan mennyi memóriára is van szükségünk, a bináris állomány fejlécéből olvashatjuk ki, ami a szakaszok, illetve az egész program méretét tartalmazza. Ezután következhet a memóriafoglalás. A hívó program memóriáját akár fel is szabadíthatjuk. A következő lépés a kód betöltése és a tényleges jogosultságok beállítása (ellenőrizni kell a `SETUID` és `SETGID` biteket). A folyamattáblába való beírás csak ezt követően mehet végbe. Utolsó lépésként a rendszermagot meg kell kérnünk a folyamat futtatására.

Élőhalottak a memóriában?

Minden folyamat számára eljön egyszer a vég. Ez a vég beköszönhet egyrészt egy jel képében, amikor is a program felszólítást kap a memóriából való távozásra. Másrészt úgy is vége szakadhat létezésének, ha egyszerűen az összes feladatát elvégezte. Ha egy gyermekfolyamat befejezte munkáját, akkor meg kell hívnia az `exit` nevű rendszerhívást. Ennek a rendszerhívásnak egy értéket is át kell adnia, mégpedig egy 0 és 255 közé eső számot. Ez az úgynevezett kilépési helyzet (exit-status), ami arra szolgál, hogy a szülőfolyamat értesülhessen róla, hogy a gyermeke milyen eredménnyel végezte el feladatát. (Ha például a gyermek 0-s státusszal tér vissza, akkor sikeresen, ha valami mással, akkor sikertelenül.)
A gyermekfolyamat végleges megszűnéséhez azonban egy második lépésre is szükség van, mégpedig arra, hogy a szülő meghívja a `wait` rendszerhívást. Ez arra szolgál, hogy a szülő várakozhasson arra, hogy a gyermeke meghívja az EXIT-et, és utána visszakaphassa annak kilépési helyzetét.
Amíg azonban a szülő nem hívja meg a `wait`-et (de a gyermek már végrehajtotta az `exit` rendszerhívást), addig a gyermekfolyamat úgynevezett zombiállapotba kerül. Ez amolyan átmenet az „élet és a halál” között, azaz már visszaadta a számára lefoglalt memóriaterületet és ütemezésre sem kerül, viszont a folyamattáblában továbbra is jelen van. Ez az állapot általában csak rövid ideig tart. Ha azonban valamilyen okból kifolyólag a szülőfolyamat még azelőtt megszűnik, hogy kiadta volna a `wait`-et, a gyermekfolyamat sosem lesz képes kikerülni a zombiság csapdájából.
Hogy ne kísérthessenek mindenféle élőhalott folyamatok a folyamattáblában, egy szülőfolyamat megszüntetésekor bizonyos intézkedések megtételére van szükség – szülőt kell találnunk az árván maradt gyerekfolyamatok számára.
A legkézenfekvőbb „apajelölt” az `init` – ami az egész folyamatfa gyökerét képezi –, mivel ez a folyamat sosem fog megszűnni. Az `exec` rendszerhívást a gyermek hajtja végre, így a szülőfolyamat tovább dolgozhat. Igaz, ebben a példában a szülő csak annyit tesz, hogy vár a gyermekfolyamat befejeződésére.
(Az `init`-ről már volt korábban szó, amikor is a `boot` műveleteket tárgyaltuk. Ez egy olyan egyszerű program, amit maga a rendszermag indít el, miután betöltődött a memóriába és befűzte a gyökérlemezrészt. Az `init`-nek egyetlenegy feladata van: minden terminál számára elindítja a login nevű bejelentkező programot. Ezután blokkolt állapotba kerül, és így is marad, amíg magát a rendszert le nem állítjuk – pontosabban az `init` sosem hajt végre `exit` rendszerhívást, de azért időnként meghív egy `wait`-et, hogy az árván maradt zombifolyamatoktól megszabaduljunk.)
A következő részben utolsó nagy témakörünk kivesézését kezdjük meg, mégpedig a fájlrendszerét. Ez egy nagyon összetett alrendszer, ugyanis nemcsak hatékonyan és biztonságosan kell tárolnia a lemezen lévő adatokat, de a felhasználóhitelesítés kérdése is kulcsszerepet kap az esetében; mindemellett a memóriakezelővel is barátkoznia kell.

Garzó András (garzoand@interware.hu)
Körülbelül három éve foglalkozik Linux- és más Unix-rendszerekkel. Legjobban az operációs rendszerek lelkivilága érdekli, de nyitott egyéniség. Kedvenc étele a palacsinta, és van egy Richard nevű macskája. Minden észrevételt, megjegyzést, levelet szívesen fogad.

Én, a robot

Önműködő webes ügynök Asimov módra.

A Web életünk részévé vált. A böngészés nemcsak gyors és egyszerű módja az információszerzésnek, hanem gyakran az egyedüli is. Ugyanakkor a Weben található ismeretanyag hatalmas mérete és a mindennapi feladatok nyomasztó sokasága miatt az adatszerzést magát jó lenne önműködővé tenni. Ezt az igényt remekül kielégítené Asimov beszélő robotja, amit önálló munkákkal bízhatnánk meg. Amíg azonban nem írunk 2058-at, marad a Perl.
Ez a cikk egy olyan robot elkészítéséről szól, ami meglátogat egy keresőmotort, végrehajtja a keresést, a kapott eredményből kiszűri a hivatkozásokat, és kiírja őket a képernyőre. A bámulatosan egyszerű példán keresztül bemutatom az LWP, URI, illetve HTML-modulok használatát. Ebből kiindulva már összetett űrlapok kitöltését végző ügynökök is könnyedén írhatók.

Ha keresés, akkor Google

A ➲ http://www.google.co.hu/-t fogjuk használni a kereséshez. Ahhoz, hogy megkapjuk a keresés eredményeit tartalmazó oldalt, a robotnak nem kell meglátogatnia a főoldalt. Vessünk csak egy pillantást a forrásra:

```
<form action="/search" name=f>
```

Majd lejjebb:

```
<input maxLength=256 size=55 name=q value="">
```

Nagyjából ennyiből áll az egész űrlap. A fejlécben található JavaScript még beszúr néhány rejtett mezőt, amelyek a karakterkódolást és az ehhez hasonló feladatokat határozzák meg. Ezek most a mi szempontunkból nem érdekesek. Ha a `<form>` elemnek nincs `method` értéke, akkor alapértelmezésben `GET`-et használ, vagyis az űrlap adatai a címben fognak szerepelni. Mivel a szövegbeviteli mező neve `q`, például a kutya szóra való rákeresés így történik:

```
http://www.google.co.hu/search?q=kutya
```

A robotnak tehát első lépésben le kell töltenie a fenti címről a HTML-állományt. Ebben segít az LWP. Debian alatt a libwww-perl csomag része, de természetesen a CPAN-ról is le lehet tölteni.

Az LWP használata

Oldalakon keresztül írhatnék az LWP korlátlan lehetőségeiről, most azonban csak néhány alapfogást mutatok be. Az LWP egy objektumközpontú felület mögé rejti a TCP/IP-s kapcsolatfelvételt, a HTTP-protokollt és a hibakezelést. A programozónak egy oldal eléréséhez alig kell többet tudnia annak címénél. Először létre kell hozni egy úgynevezett felhasználói ügynököt (UserAgent). Ennek az objektumnak a `request` elemfüggvényét kell egy HTTP-kéréshez meghívni. A függvény egy kérésobjektumot vár értékként, amit szintén egy egyszerű `new` utasítással hozunk létre. A visszatérési érték ugyancsak objektum, aminek különböző tulajdonságai tartalmazzák a válasz HTTP-kódját, annak szöveges jelentését stb.

Az URI használata

A liburi-perl része az URI-modul. Segítségével URI-kkal (Uniform Resource Identifier) kapcsolatos műveleteket végezhetünk. Egyszerű objektumközpontú felülettel bír, csakúgy, mint az LWP. Nekünk most azért lesz rá szükségünk, mert a keresési kifejezésünket (a fenti esetben a kutyát) a felhasználó határozza meg. Ezért szükségünk van arra, hogy az ékezetes betűket, szóközöket és egyéb hasonló különleges karaktereket szabványos, címben is használható formára hozzuk. Például egy címben nem szerepelhet szóköz, ezért a %20 kifejezés helyettesíti. E feladatunkat az URI-modullal könnyedén megoldhatjuk. Mindössze létrehozunk egy URI objektumot, majd az `as_string` elemfüggvényt meghívva megkapjuk a kívánt szabványos címet.

A puding próbája

Eddigi ismereteinket összegezve lássunk egy függvényt, ami az értékként megadott szövegre keres rá a ➲ http://www.google.co.hu-n (lásd az *1. listát*).
Fontos megjegyezni, hogy a `$ua->request` függvény meghívásáig semmilyen kapcsolattartás nem zajlik a hálózaton. Elsőként létrehozzuk a kívánt objektumokat, majd egy függvényhívással felvesszük a kapcsolatot a kiszolgálóval, elküldjük a kérést, megkapjuk a választ, és lezárjuk a kapcsolatot. A függ-

```
use LWP::UserAgent;
use URI;

sub search {
    my $page =
       'http://www.google.co.hu/search?q=';
# amit meglÆtogatunk

    my $ua = LWP::UserAgent->new;
# a UserAgent objektum lØtrehozÆsa
    $ua->agent("Mozilla/5.0");
# kicsi f llentØs: azt mondom, Mozilla vagyok

    my $q = URI->new(shift);
# az ØrtØkbıl egy URI objektum

    my $req = HTTP::Request->new( GET =>
       $page . $q->as_string );
# egy kØrØsobjektum

    my $res = $ua->request($req);
# a kØrØs vØgrehajtÆsa

    die $res->message . "\n"
        if $res->is_error;
# ha nem siker lt
    return $res->content; # ha siker lt
}
```

vényt egyszerűen kipróbálhatod, ha a program végére beszúrod a következő sort:

```
print search("kutya");
```

Ekkor a program lefutása után a képernyőn a kutya szóra történő keresés eredményének HTML-oldalát látod.

Szűrjük az eredményt!

Most jön a feladat oroszlánrésze. Meg kell keresni az oldalban az eredmények hivatkozásait. Jelen esetben szabályos kifejezéseket is használhatnánk, mivel a szűrési feltételek viszonylag egyszerűek. Viszont egy összetettebb HTML-oldal esetén a mintaillesztés már nem jelent megoldást, ezért lássuk, mire képes a `HTML::Parser`. Nem nehéz kitalálni, hogy melyik csomag rejtheti a számunkra értékes Perl-modult: a `libhtml-parser-perl`. Az utóbbi csomag több modul gyűjteménye, mi azonban most csak a legfontosabbal foglalkozunk.

A HTML::Parser használata

A modul használatához először létre kell hoznunk egy `HTML::Parser`-objektumot. Mivel a modul nagy múltra tekint vissza, a megfelelőség megtartása érdekében két változatát is elérhetjük a `new` függvény segítségével. A legújabb, azaz a hármas változatot fogjuk használni – ezt azért fontos hangsúlyozni, mert nem működik együtt a korábbi, kettes változattal. Egy HTML-oldal tényleges vizsgálata során az értelmező minden egyes elemnél megáll. Ha olyan elemet talál, amihez eseménykezelő tartozik, meghívja a megadott függvényt, ha nem, az alapértelmezett eljárás szerint jár el. Ilyen elemek az egyszerű szöveg, a nyitóelemek (`<table>`), a záróelemek (`</table>`), az oldalak elején gyakori meghatározások (`<!DOCTYPE ...>`), a megjegyzések (`<!-- ... -->`) stb. Az objektum létrehozása után a `handler` tagfüggvény segítségével adhatunk meg különböző eseménykezelőket. Ezeknek a megadását követően meghívhatjuk a `parse` függvényt, értékként átadva a HTML-oldalt. Ami igazán kellemes a `HTML::Parser` használatában, az az, hogy egy eseménykezelő függvény is létrehozhat, törölhet, illetve módosíthat más eseménykezelőket, így nagy bonyolultságú elemzések is végezhetők.

Szűrés a Google esetében

Vizsgáljuk most meg a Google találati oldalát! A kutya szóra történő keresés eredményének az a részlete, ahol az első találat szerepel:

```
<p class=g><a href=http://www.kutya.net/>
```

A fejléc `<style>` eleméből egyértelműen látható, hogy az oldal CSS-t használ. Ha az egész HTML-dokumentumot alaposan átböngésszük, rájöhetünk, hogy minden találat előtt szerepel egy `<p>` elem, ami a megfelelő formázás elérése érdekében „g” osztályú. Sőt csak a találatok előtt szerepel „g” osztályú `<p>` elem. Ez azért fontos, mert rengeteg hivatkozás van az oldalon, olyanok is, amelyek nem találatok a keresésünkre. Ez a `<p>` elem viszont tökéletes szűrési feltétel. Tehát egy olyan eseménykezelőt kell írni, ami a nyitóelemeket vizsgálja. Ha `<p class=g>`-t talál, akkor kicseréli a nyitóelemek eseménykezelőjét, vagyis saját magát egy újra. Az új eseménykezelő az `<a>` elemeket keresi, és az első találatnál kiírja a `href` értéket, majd visszacseréli magát a régi eseménykezelőre. Ez elsőre talán kicsit nyakatekerten hangzik, de a programkód világos és érthető lesz.

Még egy próba

Ezek után lássuk azt a függvényt, ami egy Google találati oldalból kiszűri az eredmények hivatkozásait. A függvény értéke a HTML-oldal (lásd a 2. listát; 48 CD Magazin/Robot könyvtár). Látható, hogy a `handler` függvény első értékében az eseményhez tartozó függvényt a címével adtam meg. Lehetőség van egyszerűen a nevével hivatkozni rá, én azonban ezt nem tartom helyes programozói módszernek. A `handler` második értéke egy szövegfüzér, ami tartalmazza a eseménykezelő meghívása esetén a megadandó értékeket. Ha például az értelmező a következő elembe fut bele:

```
<p class=g>
```

akkor a `start4href` függvényt három értékkel hívja meg. Az első egy egyszerű „p” szövegfüzér. A második egy asszociatív tömb címe, aminek egyetlen kulcs-érték párja van, nevezetesen a `class-g`. Harmadik, egyben utolsó értéke maga az objektum (a fenti példában a `$p` változó tartalma). Először a `start4all` függvény a nyitóelemek eseménykezelője. Ez egészen addig van így, amíg nem talál egy `<p class=g>`-t, amikor is a nyitóelemek új eseménykezelője a `start4href`. Ez az első `<a>` találatnál kiírja a `href` értékét, majd ismét a `start4all`-t teszi meg eseménykezelőnek.

A kész program

Ahhoz, hogy a fenti két függvényből működő alkalmazást varázsoljunk, csupán egymás után meg kell hívnunk őket:

```
my $content = search(shift);
filter($content);
```

Egyszerűen átadhatod parancssorban a keresési kifejezést, és a program máris tíz találat hivatkozásával gazdagítja a képernyődet. Azok számára, akik nem szeretnek sokat gépelni, a program a CD-mellékleten is megtalálható.

Felhívás egy táncra

Szeretném előre leszögezni, hogy egy ilyen robot nem arra való, hogy százezerszer leadja a szavazatodat kedvenc vagy éppen gyűlölt figurádra valamelyik valóságshowból. Sem arra, hogy a történelemtanárod levélcímét felhasználva hatszáz pornóoldalon jegyeztesd be az illetőt, és eláraszd velük a postafiókját. Ezekkel a támadásokkal szemben meglehetősen kevés mód van a védekezésre. Éppen ezért övön aluli ütésnek számít kihasználni az ehhez hasonló lehetőségeket. Hiszem, hogy ennek a folyóiratnak a közönsége van olyan érett, hogy ha választhat a pusztítás és az építés között, akkor az utóbbi mellé áll. Címemre várom az összes ötletes, új robotot. Az enyémen is fejleszthetsz, ha kedved támad, hiszen egyelőre csak az első tíz találatot írja ki. Sok szerencsét a kísérletezgetéshez!

A cikkhez tartozó listák megtalálhatóak a 48. CD Magazin/Robot könyvtárában.

Fülöp Balázs (xut@freemail.hu)
18 éves, imádja a Túró Rudit, a Debian Linuxot és a teheneket. Az ELTE Radnóti Miklós Gyakorlóiskola tanulója immár ötödik éve. Kedvenc írója Slawomir Mro□ek. Leginkább a számítógépes hálózatok biztonsága érdekli.

Video- és audiofolyam házilag

Az ffmpeg egy teljes programcsomag, ami minden eszközt tartalmaz video- és audiokódoláshoz, -átalakításhoz, -lejátszáshoz és adatfolyam-szolgáltatáshoz (stream).

Mégis, mire használható? Webkamerával élő vagy előre rögzített videoképet lehet sugározni az Interneten keresztül, szinte tetszőleges formátumban és minőségben, mindezt akár hanggal együtt. Az `ffmpeg` két fő részből áll: az egyik az ffmpeg nevű program, ami a kódolást, az átalakítást végzi. Az ffmpeg álltal támogatott formátumok között megtalálható többek között az mpeg, az mpeg-video, az mp2, az ogg, az rm, az ra, az mpjpeg, az jpeg, az asf, az swf, avi és a master. (Az egyes formátumokhoz tartozó magyarázatok az 1. listán, 48. CD Magazin/ffmpeg könyvtárban, találhatóak). Ezenkívül képes Video4Linux eszközről érkező jelet rögzíteni. A másik rész pedig az `ffserver`, ami az adatfolyamok kezeléséért felelős. A GNU LGPL felhasználási szerződés feltételeinek betartásával bárki használhatja és módosíthatja ezeket a programokat. Ez a felhasználási szerződés főleg abban különbözik a GNU GPL-től, hogyha valaki módosítja, kötelezően értesíteni kell a változásokról az alkotókat. A program fejlesztése alapvetően Linux alatt történik, de lefordítható más Unix-, illetve Windows-rendszerekre.

Telepítés

A programcsomag forrásban a fejlesztők hivatalos weboldaláról, a ➲ http://ffmpeg.sourceforge.net címről tölthető le. Fordítása a szokásos módon a következő parancsokkal történik:

```
./configure
&& make
```

majd rendszergazdaként az alábbi utasítást adjuk ki:

```
make install
```

Mielőtt mindezt megtennénk, szükség lehet néhány függvénykönyvtárra, hogy minél több tudással ruházzuk fel, például szükség lehet egy MP3-enkóderre. Én a LAME MP3-kódolót választottam (mivel ez jól együttműködik vele), amit a ➲ http://lame.sourceforge.net/ címről lehet beszerezni. Fordítása szintén a szokásos parancsokkal történik:

```
./configure && make && make install
```

Telepítettem az OggVorbis kódolót is, de ezt már csomagból. Debian GNU/Linux használata esetén adjuk ki a következő parancsot:

```
apt-get install libvorbis-dev
```

További függvénykönyvtárakat is képes használni, ezek közé tartozik például az ***Imlib2***, a ***a52***, a ***zlib*** és a ***gprof***. Ezek mindegyike Debian alatt gyorsan, fájdalommentesen telepíthető. Ha már mindent feltettünk, és használatba is szeretnénk venni az ffmpeg-et, a következőhöz hasonló parancsot kell kiadni:

```
./configure --enable-mp3lame --enable-vorbis
➥--enable-gprof --enable-a52bin
```

Amennyiben hiba nélkül futott le a fordítás, a *2. listán* látható eredményt kell kapnunk. Ezt követően jöhet is a fordítás és a telepítés:

```
make && su root -c "make install"
```

Videokiszolgáló

A videokiszolgáló használatba vételéhez elsőként az ***ffserver.conf*** felépítését kell megértenünk. Ahogy a *3. listán* látható, először néhány általános hatókörrel bíró jellemzőt kell megadni, például a kaput (portot), azután, hogy melyik IP-címen figyeljen, továbbá az ügyfelek legnagyobb számát, a használható legnagyobb sávszélességet stb.
A beállítófájl további része kétféle blokkból épül fel. Az egyik az úgynevezett ***feed***, a másik pedig a ***stream***. Nagyon leegyszerűsítve a dolgot a feed blokk(ok) szerint feltöltött adatokat a stream blokk(ok)ban meghatározott tulajdonságoknak megfelelő formátumban lehet letölteni. Bármelyik stream bármelyik feedet használhatja. Természetesen a fel-, illetve letöltés bárhonnan történhet, csak az adott IP-címet engedélyezni kell a hozzáférési listákban (ACL). Mindebből következik, hogy legalább egy feed és egy stream blokknak kell lennie.

```
Install prefix   /usr/local
Source path      /usr/src/media/ffmpeg-0.4.6
C compiler       gcc
make              make
CPU               x86
Big Endian       no
MMX enabled      yes
gprof enabled    yes
zlib enabled     yes
mp3lame enabled  yes
vorbis enabled   yes
a52 support      yes
a52 dlopened     yes
Video hooking    yes
Imlib2 support   yes
Creating config.mak and config.h
```

```
Port 8090
BindAddress 0.0.0.0
MaxClients 10
MaxBandwidth 5000
CustomLog /var/log/ffserver/access.log
#NoDaemon
```

```
<Feed camera1.ffm>
File /tmp/camera1.ffm
FileMaxSize 200K
#Launch
ACL allow 127.0.0.1
ACL allow 10.0.0.0 10.0.0.255
</Feed>
```

A feed blokkot a *4. listán* látható módon kell megadni. Ez határozza meg, hogy a kiszolgáló honnan fogja kapni az adatokat, és milyen jellemzőkkel. Adatokat feltölteni az ffmpeg programmal lehet, de erről majd később szólok. A streamblokk meghatározza a szolgáltatott video-, illetve audio-adatfolyamok nevét és egyéb tulajdonságait (5. lista, lásd 48. CD Magazin/ffmpeg könyvtár). A formátum egészen „különleges” is lehet, például jpg vagy akár Flash-animáció. A legjobb képet mindenképpen az MPEG4 adja (a sávszélességhez képest), de sajnos ezt csak MPlayerrel sikerült lejátszanom. A Windows Media Player képtelen az adatfolyamként kapott AVI-t értelmezni.
Ezek után a kiszolgálót a következőképpen indíthatjuk:

```
ffserver -f ffserver.conf
```

Egy beállítási példafájlt a forrásban is lehet találni. Az ffserver egy egyszerű webkiszolgálót biztosít számunkra, amin keresztül többek között az adatfolyamok is elérhetők. Ez a webkiszolgáló alapesetben a 8090-es kapun érhető el. Ezenkívül egy ***stat.html***-t és egy ***index.html***-t szolgáltat, amik természetesen ugyancsak felülírhatók. Az ***index.html*** a hivatalos weboldalra (➲ http://ffmpeg.sourceforge.net/) irányít át, a ***stat.html*** pedig adatokat szolgáltat a kiszolgáló állapotáról. Ilyen adatok például az egyes adatfolyamok tulajdonságai, az adatforgalom és a pillanatnyi kapcsolatok száma.

A kódoló

Amennyiben a kiszolgáló fut, máris indítható a kódoló:

```
ffmpeg http://localhost:8090/camera1.ffm
```

Ekkor a 6. listán láthatóhoz hasonló eredmény kell kapnunk. Ebben az esetben az adatot a */dev/video* eszközről veszi, ami lehet például egy webkamera vagy akár egy tévékártya is. Megadható más eszköz is, például: `-vd /dev/video10` vagy egy fájl is a `-i` kapcsolóval.
Természetesen a videokiszolgáló nélkül is használható a fájlok kódolására, illetve dekódolására, ugyanis megfelelő függvénykönyvtárakkal kezelni tudja az MP3, ac3, vorbis hangkódolókat és az MPEG1, MPEG2, MPEG4 (divx4, illetve divx5), MJPEG, WMP7, Huff YUV videokódolókat.
További hasznos kapcsolók a teljesség igénye nélkül:
`-t`: a felvétel időtartalma másodpercekben.
`-b`: a kódolás minőségét Kbit/s egységekben lehet megadni. Ha a videó minőségén nem javít, nem emeli a megadott mértékig.
`-s`: a kép méretének megadása, például: szélesség×magasság (352×288).
`-croptop`, `-cropbottom`, `-cropleft`, `-cropright`: képernyőpontokban megadva az az érték, hogy sorrendben fentről, lentről, balról és jobbról mennyit vágjon le a képből.
`-intra`: csak intra képkockák használata, tehát nem használ kulcsképkockákat.

```
testgep:~$ ffmpeg -vd /dev/video0
  ➥http://localhost:8090/camera1.ffm
Input #0, video_grab_device, from
  ➥'/dev/video0':
  Stream #0.0: Video: rawvideo, yuv422,
  ➥352x288, 15.00 fps, 800 kb/s
Output #0, ffm, to
  ➥'http://localhost:8090/camera1.ffm':
  Stream #0.0: Video: msmpeg4, 352x288,
  ➥15.00 fps, q=3-31, 768 kb/s
  Stream #0.1: Video: mjpeg, 352x288, 1.00
  ➥fps, q=3-31, 64 kb/s
  Stream #0.2: Video: msmpeg4, 352x288, 8.00
  ➥fps, q=3-31, 96 kb/s
Stream mapping:
  Stream #0.0 -> #0.0
  Stream #0.0 -> #0.1
  Stream #0.2 -> #0.2
Press [q] to stop encoding
frame=   17 q=3.0 q=11.0 q=12.0 size=
  ➥88kB time=1.1 bitrate= 636.1kbits/s
```

`-vcodec`, `-acodec`: video-, illetve audiokódoló. Ha csak a `copy`-t adjuk meg, akkor módosítás nélkül hagyja.
`-benchmark`: a kódolás végén tájékoztat a kódolás sebességéről.

A kódoló képes a ***.vob*** fájlok (DVD) értelmezésére is, így tetszőleges program által ismert formátumra átalakíthatóak. Meg voltam elégedve a teljesítményével, mert egy 533 MHz-es Celeron processzort tartalmazó gépben egy webkamera képének a feldolgozása 30–40 százaléknál jobban nem terhelte meg a gépet. Nagyon hasznos és jól használható eszközről van szó, ezért több projekt használja, például a magyar fejlesztésű MPlayer, a Motion mozgásérzékelő és még számtalan videolejátszó, illetve -feldolgozó program. Eddigi használata során semmilyen gondom nem akadt vele, pedig ez még messze nem a végleges változat – a Miskolci Egyetem közreműködésével hamarosan elkészül az IPv6-ot is támogató változata is.

A cikkhez tartozó listák megtalálhatóak a 48. CD Magazin/ffmpeg könyvtárában.

Kolcza Péter (kpeter@sysconfig.hu)
Imádja a South Parkot. A Miskolci Egyetem informatika szakos hallgatója. Elvakult Linux-rajongó. Ha egyetemi elfoglaltságai engedik, Linuxszal és rendszerépítéssel foglalkozik.

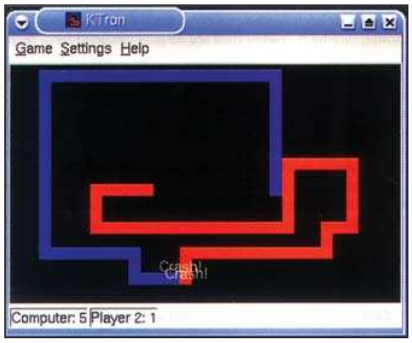
KAPCSOLÓDÓ CÍMEK

➲ http://ffmpeg.sourceforge.net
➲ http://lame.sourceforge.net
➲ http://www.mplayerhq.hu
➲ http://www.xiph.org/ogg/vorbis
➲ http://www.gnu.org/licenses/lgpl.html

Csaták a számítógép belsejében

A Tronban játszó Jeff Bridges ugyan sosem leszel, de azért nem ártana menőbb cuccokban járnod…

Hogy érted azt, hogy ki van a számítógép belsejében? Senki nincs a számítógépben, François. Ó, már értem! Azt hiszem, te félreértetted e havi témánk ötletét, „A rendszer belső világa” címet. Hiszen a tudományos-fantasztikus témák körébe tartozik minden olyan elképzelés, ami szerint a gépben a Linux-rendszermagban keringő biteken és bájtokon kívül bármi is lenne, mon ami. Qu'est-ce que tu dis? Nem, természetesen nem. A számítógépen belül nincs igazán hely, legalábbis a mi világunkhoz hasonló. Habár…
Quoi? Ne haragudj, mon ami, csak eszembe jutott egy régi mozifilm, a Tron, amiben tényleg létezett egy számítógépen belüli világ, ebben a programok a központi vezérlőprogrammal, az MCP-vel szemben harcoltak a felhasználókért. François, mit nézel annyira?
Áh, mes amis! Isten hozott titeket Chez Marcelnél, a fenséges Linux-konyha és kitűnő borok otthonában. Foglaljatok helyet, amíg François-t borért szalajtom. Van egy kis kanadai francia tourtière (húspástétom) is a háznál, amit fel fogunk szolgálni. François, siess a pincébe, és hozd a bort! Az 1997-es Napa Valley Cabernet Sauvignon remek ital lesz ma estére.
Örülök, hogy itt vagytok, mes amis. François-nak támadt egy olyan elgondolása, hogy „a rendszer belső világa” valami olyasmit takar a Linuxon belül, amiben élő ember vesz részt. Erről a Disney 1982-es Tron című filmje is eszembe jut, amiben a számítógép belsejében programok vívják gladiátori küzdelmeiket a felhasználóikért. Ebben a virtuális világban a halálos sportok egyike a fénymotorok versenye volt. Az ellenfelek olyan motorokon versenyeztek, amelyek a nyomukban fényfalat emeltek. A motoroknak folyamatosan haladniuk kellett, miközben mindegyik versenyző próbálta elkerülni az ellenfél falait, és igyekezett úgy húzni a sajátját, hogy az ellenfelek ne menekülhessenek meg az ütközéstől. Az utolsó talpon maradó program lett a győztes.
François, csakhogy megjöttél! Tölts, kérlek a vendégeknek, és siess vissza a konyhába azért a tourtière-ért. Bár a Tron egy kicsit elavultnak tűnik napjainkban, mégis, a fénymotorok küzdelme volt annak a megszámlálhatatlan videojátéknak az ötletadója, ami a folyamatosan növekedő fénycsóvák ötletén alapul, amelyeket ugye mindenáron el kell kerülni. Ez az ötlet ma is él, és több nyílt forrású fejlesztés alapjaként születik újjá. Kezdjük egy olyan fénymotorjátékkal, amit nagy valószínűséggel már a rendszerünkön tudhatunk. A KDE játékgyűjteményének részeként találhatjuk meg ***Matthias Kiefer*** Ktron nevű, egyszerű, de jól működő programját. A ***K*** menüben, az ***arcade games*** (többszereplős játékok) menüpont alatt találjuk, a parancs neve `ktron`. A játék elindítása után két négyzetet, egy kéket és egy pirosat láthatunk a képernyő közepe táján. Az indításhoz nyomjuk meg a bal vagy jobb kurzorbillentyűt, és már indul is a játék (*1. kép*).



1. kép KTron – a KDE saját fénymotorjátéka

A játékmenet megváltoztatásához a menüsor ***Settings*** (beállítások) pontját kell kiválasztanunk, és máris rendelkezésünkre állnak a különféle módosítási lehetőségek. Ha a játék egy kicsit gyorsnak tűnik, csökkentsük a sebességet, vagy ellenkezőleg, gyorsítsuk, ha el találnánk aludni a kerekek fölött. A motorok nyomának méretét vagy a beállított színeket is megváltoztathatjuk. Már ez a kis játék is rengeteg örömet tud szerezni, különösen, ha megváltoztatjuk a küzdőtér méretét; de ha valóban érezni akarjuk egy fénymotoros csata hangulatát, be kell lépnünk a háromdimenziós világba. Most, mes amis, tényleg bejutunk a számítógép belsejébe.

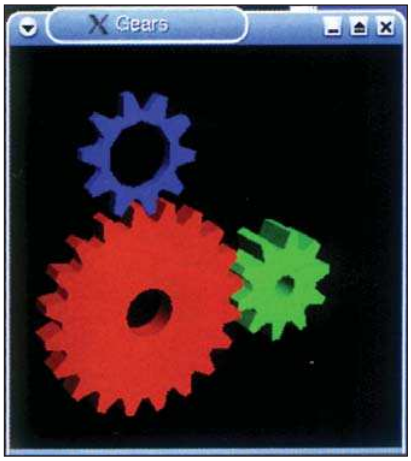
Egy 3D-gyorsítóval rendelkező videokártyára, valamint az OpenGL vagy Mesa 3D programkönyvtárakra lesz szükségünk.
Honnan tudhatjuk, hogy rendszerünk készen áll a kalandra? A 3D-támogatás gyors ellenőrzési módszere az alábbi parancs:

```
glxinfo | grep rendering
```

A rendszernek erre a következő választ kell adnia:

```
direct rendering: Yes
```

Egy másik igen szórakoztató próba végrehajtásához a Mesa-demos programcsomag gears nevű programját kell lefuttatnunk. Ehhez egyszerűen a `gears` parancsot kell egy `xterm` ablakban kiadnunk. Ennek a kemény munkának a jutalma három pörgő fogaskerék lesz a képernyőnkön (*2. kép*).



2. kép Háromdimenziós fogaskerekek: a Mesa bemutatója

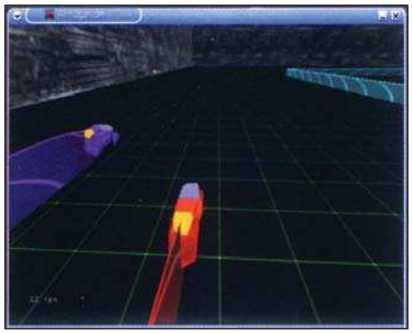
Ne nagyon hagyjuk magunkat a látványtól elvarázsolni, inkább kortyoljunk bele a borunkba, és térjünk vissza a terminálablakunkhoz. Itt a videokártyánk 3D-teljesítményére vonatkozó statisztikát láthatjuk:

```
1778 frames in 5.001 seconds
➥= 355.529 FPS
```

Ezt az eredményt az nVvidia GeForce2 kártyával felszerelt próbarendszerem hozta létre, és igazából nagyon jó teljesítményt jelent. Ezzel szemben a kis noteszgépem, ami teljesen nélkülözi a gépi gyorsítást, a következő eredményt kapta:

```
312 frames in 5.004 seconds
➥= 62.350 FPS
```

Ha egyáltalán nincs a kártyánkon beépített gyorsító, a program akkor sem áll le, egyszerűen csak rettentő lassan fog futni. Most, hogy ezt az akadályt leküzdöttük, térjünk vissza a fénymotorok háromdimenziós világába ***Manuel Moos*** Armagetron nevű programjának a segítségével (➲ http://armagetron.sourceforge.net). Ahogy azt Manuel honlapja is javasolja: gondoljunk az Armageddonra, csak „tr”-rel a hosszú „d” helyett. A honlapon könnyen találhatunk előre fordított bináris állományokat, de a forráskód is rendelkezésünkre áll.



3. kép Háromdimenziós fénymotorjáték – az Armagetron

4. kép A GLTron – olyan élmény, hogy szinte a számítógép belsejében érezzük magunkat

Amikor először próbáltam ki a programot, rögvest egyenesen a falnak hajtottam. Még néhányszor eljátszottam ugyanezt, amikor eszembe jutott, hogy talán nem ártana a kormányzás billentyűit is beállítanom magamnak. A menüből a ***Player settings*** (a játékos beállításai) menüpontot (mindegy, milyen számmal szerepelünk), és itt az ***Input Configuration*** (az irányítás beállítása) feliratot kell kiválasztanunk. Lépkedjünk a lefelé nyíllal a ***turn left*** (fordulás balra) feliratig, és nyomjuk meg az ENTER billentyűt. Most azt a gombot kell megnyomni, amit a balra kanyarodáshoz szeretnénk használni. Tudom, hogy furán hangzik, de én erre a célra a balra nyilat választottam. Ismételjük meg ezt a folyamatot a jobbra forduláshoz (***turn right***) is, majd minden máshoz, amit a küzdelem alatt használni szeretnénk – például arra is lehetőségünk van, hogy egy gombnyomás segítségével gúnyolódjunk az ellenfeleinkkel. Ezután lépjünk ki ebből a menüből és a korábbiakból is mindaddig, amíg vissza nem jutunk a főmenübe.

Most a ***Game*** (játék) menüből válasszuk a ***Start Game*** (a játék indítása) menüpontot. Az Armagetron nagyszerű fénymotoros program, amiben egy mindenütt jelen lévő és minden mozgást követő fej feletti kamera szemszögéből követhetjük az eseményeket, és néha szédítő nézőpontváltásokat tapasztalhatunk a hirtelen bekövetkező kilencven fokos fordulatoknál (*3. kép*).

Az Armagetron hálózati játékra is alkalmas, összesen 16 gépet tud kezelni, amelyek mindegyikén négy versenyző futhat egyszerre.

A program hamar függőséget okozhat, ezért szükségszerűen rendelkezik azzal a nélkülözhetetlen szolgáltatással, amit főnökgombnak szokás nevezni. Röviden arról van szó, hogyha esetleg a munkahelyi hálózatunkon folytatjuk ádáz harcunkat az ellenfelekkel, és hirtelen közeledni látjuk a főnököt, a SHIFT-ESC billentyűkombináció hatására azonnal kiléphetünk a programból, ami nyomban el is tűnik a képernyőről.

Az Armagetron nagyszerű játék, de a fotorealisztikus élményhez ***Andreas Umbach*** GLTron nevű programjára van szükségünk. Mint várható, ezért a látványért keményen meg is kell fizetnünk, legalábbis a számítógép teljesítményét illetően, de ha már rendelkezünk a megfelelő masinával, biztosan nem fogunk csalódni. A bináris állományok felkutatása nem jelenthet gondot (az ➲ http://rpmfind.net címen találhatunk RPM-csomagokat), a forrás pedig bármikor elérhető a ➲ http://www.gltron.org címen.

A program (megfelelő 3D-teljesítményt feltételezve) meglehetősen gyors, lenyűgöző grafikával és egy olyan játékmóddal rendelkezik, ami biztosan megemeli a pulzusszámunkat.

A beállításokat három fő csoportban tehetjük meg – ***game*** (játék), ***video*** (kép), ***audio*** (hang) –, és ezek némelyike alkalmas arra, hogy drámaian megváltoztassa a játékélményt. A GLTron mind teljes képernyős, mind pedig ablakos futtatásra alkalmas. A megjelenítést és a hangulatot kiegészítő csomagokkal (artpacks) tehetjük változatossá. A kedvencem ezek közül még mindig az alapértelmezett, de a Metaltron is belopta magát a szívembe.

Az F10 billentyűvel állítható a kamera távolsága és a nézet szöge. Nagyon érdekesnek találtam a motor mögötti kameraelhelyezést is. Ez olyan látványt nyújt, mintha magán a fénymotoron ülnénk – nagyon vad élmény.

Az alapbeállítás szerint egy kis kétdimenziós térkép látható a képernyő tetején. Ha kezdjük elbízni magunkat, érdemes kipróbálni a kikapcsolását.

Sok szempontból a GLTron a legkidolgozottabb a három most bemutatott játék közül, viszont nem rendelkezik az Armagetron hálózatos képességeivel. Egyszerűségénél fogva viszont a Ktron bármilyen számítógépen játszható, függetlenül a videokártyába épített 3D-képességektől.

Mon Dieu, ilyen gyorsan elment az idő? Mivel később már nem játszhatunk és nem tölthetek nektek még egy pohár bort, most kell megtennem, még mielőtt François-val bezárnánk éjszakára az éttermet. François, lennél szíves utoljára teletölteni a poharakat? Egészségetekre, mes amis, a legközelebbi viszontlátásig! A votre santé! Bon appétit!

Linux Journal 2003. május, 109. szám

Marcel Gagné
(mggagne@salmar.com) Mississaguában, Ontario államban él. Ő a szerzője a Kiskapu kiadásában tavaly szeptemberben megjelent Linux-rendszerfelügyelet című (ISBN 96-9301-40) könyvnek.

KAPCSOLÓDÓ CÍMEK

Armagetron
➲ http://armagetron.sourceforge.net

GLTron ➲ http://www.gltron.org

KTron ➲ http://games.kde.org

Marcel borlapja
➲ http://www.marcelgagne.com/wine.html

Bemutatjuk a Plone-t

Vajon a Plone hozza el a Zope alapú tartalomkezelést a tömegek számára?

A Zope felülethez készített Plone nevű tartalomkezelő rendszer (Content Management System – CMS), készítői: ***Alexander Limi***, ***Alan Runyan*** és ***Vidar Andersen***. A Plone népszerűsége az elmúlt hónapokban egyre nőtt, s rengeteg új embert vonzott a Zope-közösséghez. Bár a Plone-felhasználók többnyire a saját területükön maradnak és nemigen érdeklik őket a Zope-kiszolgáló rejtelmei, a Plone igazi sikeralkalmazása a Zope-nak. És ha ez így van, az érdekes következtetésekhez vezethet, különös tekintettel arra, hogy maga a Zope is a Python sikeralkalmazásaként indult.

Mi is a Plone?

A Plone tulajdonképpen egy egyszerű CMS, ami segíti a felügyelőket abban, hogy minden egyes felhasználót külön jogosultságokkal ruházhassanak fel. A nyilvános részeket minden felhasználó elolvashatja. Bizonyos felhasználók új elemeket vihetnek fel, míg mások nyilvánossá tehetik őket a közösség számára. A CMS és egy hagyományos webhely között pontosan ez a meglévő és nyilvános közötti megkülönböztetés a legjelentősebb eltérés.
Az állandó weblapokkal ellentétben – amelyek esetében a közönség azonnal elérheti a dokumentumkönyvtárban megjelenő fájlokat – a CMS lehetővé teszi, hogy eldöntsük, mely dolgok legyenek elérhetők, és melyek nem. Továbbá egy CMS-rendszerben a korábban kiadott anyagokat bármikor visszavonhatjuk. Így ha a webhelyen megjelent egy hír, amiről kiderül, hogy hamis, fájltörlés nélkül eltüntethetjük a nyilvánosság szeme elől. A lap rendszergazdáját folyamatosan készülő napló tájékoztatja arról, hogy melyik cikk mikor készült, mikor közölték le, vonták vissza, illetve ki és milyen célból végezte el ezeket a változtatásokat.
A Plone sem mindentudó. Fő célja, hogy még szebb felületet és testreszabható webhelyet nyújtson a kisebb és közepes nagyságú weblapok felügyelői számára, akik számos hasznos, ügyes és apróságra vágynak.
Maga a Plone több különböző Zope-termékből áll, és valamennyi termék több példányban is létrehozható objektum. Bár a Plone nem közvetlenül a Zope alatt, hanem annak tartalomkezelő keretrendszerében (Content Management Framework – CMF) íródott, jó néhány objektum és API segít bennünket saját keretrendszerünk létrehozásában.
A Plone 1.0 még a cikk születése előtt, 2003 elején jelent meg, és nagymértékben függ a 2002 közepén kibocsátott CMF v1.3 rendszertől. Ahhoz hasonlóan, ahogyan az asztali alkalmazások kihasználják az operációs rendszer nyújtotta szolgáltatásokat, úgy aknázza ki a Plone (vagy bármelyik más CMF alapú CMS) a CMF által felkínált lehetőségeket. A Plone-weblapok ennek megfelelően teljes körű szövegkeresési szolgáltatással bírnak, illetve támogatják, hogy a közönség a weblap bármelyik részéhez megjegyzést fűzhessen. Ahogyan a CMF-et tökéletesítik és egyre több szolgáltatást kínál, várhatóan a Plone is fejlődik majd.

A Plone telepítése

Amennyiben a Zope már üzemel, a Plone telepítése nagyon egyszerű lesz. Ne feledjük, hogy minden Zope-alkalmazást a Zope mappa ***lib/python/Products*** könyvtárába kell elhelyezni. Ezenkívül vagy kézzel, vagy a weblapos beállítópulttal újra kell indítanunk a Zope-ot, hogy megtörténjen az új alkalmazások felvétele.
A Plone telepítése előtt kell telepítenünk a legfrissebb CMF-változatot. Töltsük le a gzippelt ***CMF-1.3*** tarfájlt a ➲ http://cmf.zope.org honlapról. Én ezt a ***/tmp*** könyvtárba helyeztem, és a következőképpen telepítettem:

```
# cd $ZOPE/lib/python/Products
# tar -zxvf /tmp/CMF-1.3.tar.gz
```

A ***CMF-1.3*** új könyvtárat készít a ***lib/python/Products*** mappában, ami számos, a CMF-fel kapcsolatos alkalmazást tartalmaz – ezekre a Zope indulásakor lesz szükség. Ezért készítsünk néhány közvetett hivatkozást a termékmappákhoz:

```
# ln -s CMF-1.3/CMFCore .
# ln -s CMF-1.3/CMFCalendar .
# ln -s CMF-1.3/CMFDefault .
# ln -s CMF-1.3/CMFTopic .
```

A CMF telepítése után már felrakhatjuk a Plone-t is. Töltsük le a Plone tarállományt a ➲ http://www.plon.org-ról (48. CD Magazin/Plone), tegyük a ***/tmp*** mappába, majd csomagoljuk ki:

```
# cd $ZOPE/lib/python/Products
# tar -zxvf /tmp/CMFPlone-1.0.tar.gz
```

Akárcsak a CMF magprogram esetében, ismét készítsünk közvetett hivatkozásokat a ***Products*** mappában, hogy induláskor a Zope észrevegye őket. Amennyiben 2.6.x előtti Zope-változatot használunk, elképzelhető, hogy a fő ***Products*** mappában egy másik programra is közvetett hivatkozást kell készítenünk.
A biztonság kedvéért ellenőrizzük a Plone leírását. Ellenőrizzük, hogy a ***CMF-1.3*** és a ***CMFPlone*** mappát, valamint tartalmukat a Zope-ot futtató felhasználó birtokolja-e. Ez a felhasználó általában a ***www*** vagy a ***zope***. A Zope futtatása ***nobody*** felhasználóként (amit régebben biztonságos módszernek tartottak) ma már nem javasolt. Ha nem a megfelelő felhasználó birtokolja a fájlokat, könnyen előfordulhat, hogy néhány furcsa tulajdonjogi és engedélyezési nehézséggel kerülünk szembe.

Plone-weblap készítése

A Zope webalapú kezelőfelület révén nagyon könnyű Plone-weblapot készíteni. Az ***Add Product*** menüből válasszuk ki a Plone-webhelyet, majd adjuk meg a következő adatokat:

- a webhely azonosítója (ez a cím része lesz),
- a webhely (miden lap tetején megjelenő) címe,
- a webhelynek külön felhasználói mappája legyen, vagy a többi Zope-laptól örökölje a felhasználókat,
- a webhely leírása,
- a webhely típusa (hagyjuk az alapértelmezett értéken).

Miután a Zope befejezte az új Plone-weblappéldány indítását, a Zope kezelőfelületének nagy lapja komoly változásokon megy keresztül. A közepén egy bemutatkozó szöveg jelenik meg, az eszköztár felül lesz, az adatok – a naptárat is beleértve – a bal és jobb oldalon látható téglalapokba kerülnek.
A Plone-weblapokhoz használt kezelőfelület egészen más, mint a Zope-alapfelület. A Zope minden felhasználónak ugyanazt jelenítené meg, és csak akkor mutatja meg a kezelőfelületet, ha a */manage* eljáráshívást hozzáfűzzük a címhez. Ezzel szemben a Plone a pillanatnyi felhasználó engedélyei szerint módosítja a kimenetet. Így a vendégek csak nézegethetik a webhelyet, a felügyelők viszont olyan táblákat is láthatnak, mint nézet, szerkesztés, tulajdonságok és állapot, valamint az összes tételt megnézhetik, ideértve a még nem nyilvánosakat is.
Szerencsére a Plone felhasználói felügyeleti felülete eléggé könnyen megérthető. A weblap módosításához az ***edit*** (szerkesztés) fülre kell kattintani. Itt böngészőnk segítségével módosíthatjuk a tartalmat, a címet, valamint az összefoglalót. A keresések eredményében is ez az összefoglaló jelenik majd meg. A Plone felhasználói felületéhez JavaScriptet használ, hogy a kevésbé szakértő felhasználók számára is könnyen kezelhető legyen. Így ha valamilyen HTML-elemre, például egy szövegmezőre vagy rádiógombra kattintunk, a Plone szerkesztőfelülete jegyzetszerűen (tooltip like) tájékoztat bennünket, mit is kell beírnunk.
A webalapú Plone-felülettel felvitt tartalom lehet egyszerű vagy html formátumú szöveg, illetve a Zope saját szöveges formátumát is használhatjuk, ami a formázási műveletekhez központozást és bekezdéseket használ.
Én többnyire ezt a strukturált szöveget szeretem használni, HTML-hez csak akkor nyúlok, ha egy lapot strukturált szöveggel már nem tudnék létrehozni.
A lap fő dokumentumát a tartalmat kiegészítő apró segédletek (portlet) veszik körül. A Plone alapértelmezés szerint rengeteg ilyen segédletet ismer. Többek között használhatjuk a cikkek listáját, események megjelenítését, dátumot kiíró naptárat, a hónap fontos eseményeit kihangsúlyozó elemet, vagy az adott weblap témájához kötődő további dokumentumok listáját (***related portlet***).
Ha új dokumentumot szeretnénk felvenni, a navigációs segédlet hivatkozására kattintva lépjünk át a tartalom nézetre (***contents view***). Itt a pillanatnyi mappa dokumentumainak a listájához jutunk. A bal felső sarokban található ***add new item*** (új elem felvétele) mezőben kiválaszthatjuk az új tartalomtípust. Nem árt tudnunk, hogy az új dokumentum azonnal létrejön az ***add new document*** gombra történő kattintás után, a tulajdonságokat és a tartalomtípust csak ezt követően módosíthatjuk.
Alapértelmezés szerint a Plone számos tartalomtípust ismer:

- A mappák segítségével webhelyünket hierarchikusan rendszerezhetjük. Hasonlóan a merevlemezhez, a statikus vagy a Zope-webhelyek mappáihoz a Plone-webhelyek is tartalmazhatnak könyvtárakat. A nyilvános mappák címe a navigációs segédletben jelenik meg.
- A Plone-weblapok leggyakoribb elemei a dokumentumok, amiket html formátumban, strukturált szövegként vagy egyszerű szövegként adhatunk meg. Legtöbbször valószínűleg új dokumentumot akarunk majd létrehozni.
- A képek szinte bármilyen formátumúak lehetnek, beleértve a jpeg, png és a gif formátumúakat is.
- A fájlok olyan elemek, amiket elérhetővé vagy letölthetővé szeretnénk tenni a nagyközönség számára, de nem tartozik hozzájuk a Plone munkáját megkönnyítő `MIME` típus. Ilyen például a QuickTime film, az audioklip és a Microsoft Office dokumentum.
- Az események (event) olyan kezdő és végdátummal bíró rövid dokumentumok, amik a naptársegédletben jelennek majd meg.
- A hírek (news) típusú rövid dokumentumok az újságsegédletben jelennek meg. Ilyen módon egyszerűen tehetünk közzé például sajtóközleményeket.

A Plone-nal együtt néhány hivatkozást is kapunk, amik közt érdekes külső címeket és előre meghatározott weblapon belüli kereséseket találunk. Egyre több új tartalomtípus születik a Plone-hoz, hogy csak két példát említsünk: már létezik webnapló (weblog) és a fotóalbum (photo album) elem is.

Dokumentumok közlése

Plone alatt alapértelmezés szerint minden új tartalom látható lesz. Ez annyit jelent, hogy aki ismeri a címet, az böngészőjével hozzáférhet a dokumentumunkhoz. Igaz, a tartalom sem a keresőben, sem a navigációs segédletben. nem jelenik meg
A dokumentum közzétételéhez kattintsunk a lap tetején található ***state*** elemre. (Tartalomnézetből egyszerre több tartalmat is közzétehetünk a lap alján található ***state*** gomb segítségével.) Innen egy másik lapra kerülünk, ahol meg kell adnunk, hogy mikorra szeretnénk időzíteni a lap megjelenését, mikor járjon le, valamint megjegyzésben megadhatjuk a dokumentum közzétételének okát. A begépelt dátumok kötelező érvényűek, azaz a nyilvános dokumentum kizárólag a kezdő és a végdátum között lesz látható. Így már napokkal vagy hetekkel a tervezett megjelenés előtt anélkül felvihetjük a dokumentumokat, hogy a megfelelő időben változtatnunk kellene az állapotukon.
A közzététel után a dokumentum azonnal kereshetővé válik, illetve azokban a könyvtárakban, amelyekben nem található ***index_html*** dokumentum, böngészhető lesz.
Egyik kedvenc Plone-képességem a tulajdonságok kezelése. Minden dokumentumhoz hozzárendelhetünk egy vagy több tulajdonságot a képernyő tetején található tulajdonságkezelővel (***properties management***). Amikor a felhasználó megtekinti a lapot, a viszony (***related***) segédlet a webhely összes olyan dokumentumát felsorolja, aminek egy vagy több tulajdonsága megegyezik a jelenlegi oldal tulajdonságaival. Ez a szolgáltatás a látogatóknak lehetővé teszi, hogy az őket esetlegesen érdeklő egyéb adatokat is könnyedén megtalálják.

Összegzés

A Zope igen hatékony alkalmazáskiszolgáló, a CMF pedig nagyon jó CMS-készítő eszköz. Figyelembe véve azonban a mindkét alkalmazással kapcsolatos meredek tanulási görbét, elképzelhető, hogy a Plone lesz a Zope igazi sikeralkalmazása, ami könnyedén telepíthető, állítható be és kezelhető, és nagyszerű képességei révén új embereket hoz majd a Zope világába.

A kapcsolódó címek, valamint a Plone megtalálhatóak a 48. CD/Magazin/Plone könyvtárában.

Linux Journal 2003. május, 109. szám

Reuven M. Lerner (➲ http:/www.lerner.co.il/atf) Nyílt forrású programokra, valamint web- és adatbázis-alkalmazásokra szakosodott tanácsadó. Könyve, a Core Perl, 2002 januárjában jelent meg a Prentice Hall gondozásában. Reuven feleségével, és lányával Izraelben, Modi'in-ben él.

Melyiket válasszam?

Azt amelyik biztosan megy Linux alatt: az Asus M2400E-t.

Egy ismerős cégénél merült fel nem is olyan régen a választás kérdése. Felvettek egy új embert egy új állásba, és rögtön azzal a nehézséggel kerültek szembe, hogy milyen notebookot is kapjon. Mivel programmérnökként sokat tartózkodik külső munkán, csak és kizárólag egy hordozható gép a megoldás. Nagy szerencséjükre óriási a kínálat, Linux-szempontból azonban a hazai helyzet változatlan. Az eladók döntő többsége ugyan már nem próbál meg a „Jó napot kívánok! Megy az a gép Linuxszal?” kérdés hallatán elmenekülni, hanem vagy egyenesen megmondja, hogy nem tudja, vagy neki minden mindegy alapon azt feleli: „Igen, naná”. Mindenki döntse el, melyik a jobb. A kérdés tehát továbbra is az, hogy megvegyem-e a nagyjából húsz százalékkal drágább hordozható gépet, amiben olyan alkatrészek sorakoznak, amelyek 99 százalékban Linux-megfelelők, vagy kockáztassak? A válasz a következő: vegyük meg a kockázatos gépet, de csak ha legfeljebb négy munkanap alatt közepes kínlódás mellett gépet lehet belőle varázsolni. Ez azonban kizárólag vállalkozó kedvű vásárlóknak ajánlható, vagy azok számára, akik – ha nem megy rajta a Linux – a másik rendszerrel használják majd. Számomra azonban ez nem elfogadható, mivel szerencsére már vagy hat éve csak Linuxot használok.

Így el is ballagtam a Sowah nagykereskedésbe, és kölcsönkértem egy új típusú noteszgépet. Itt a nem sok jóval kecsegtető Asus M2400E-t kínálták, amiben számos alaplapi SIS lapka van. Két évvel ezelőtt a nagykereskedés ajtajából visszafordultam volna, hogy ezt ki sem próbálom, mivel úgysem fog menni. Ugyanakkor két év a Linux-meghajtók piacán körülbelül öt normál évnek felel meg. Bevallom őszintén, a polcokon lévő gépeket böngészve a 10-12" képátmérőjű, 1 kg-os gépeket kerestem, hiszen szeretem mindenhova elvinni a masinámat. A kereskedő azonban felvilágosított, hogy bár az Asus gyárt ilyen gépeket is, hozzánk nem szállítják – a magyar igényeknek ugyanis a nagy LCD-vel bíró, közepes tömegű, azonban alapkiépítésben sok extrát tartalmazó gép felel meg leginkább. Az M2400E egy 1,9 kg össztömegű, 14"-os LCD-jű gép. Valóban nem nehéz, de van benne minden földi jó: DVD-olvasó, CD-újraíró, párhuzamos kapu, infrakapu, USB-kapu stb.

Egy gyors metrózás után remegő kezekkel billentyűztem be a legnagyobb linux laptopoldal oldal címét. Rövid keresgélés után meg is találtam: igen, előttem egy német srác már felküzdött egy SuSE 7.2-es változatot egy ilyen gépre. Leírásából arra következtettem, hogy nem lesz nagy ügy. Nos, aki nekikezd, annak mindenképpen ajánlom, hogy első körben egy monitorra kösse ki a képet. Ugyanis még a 2.4.20-as rendszermag sem a megfelelő kerettárazó (framebuffer) eszközvezérlőt tartalmazza a SIS videokártyához, így a Debian telepítő alja (ahol többnyire a választógombok vannak) mindig kicsúszik az LCD-ről. Rendes monitoron nézve sem tökéletes a helyzet, de ott legalább el lehet jutni odáig, hogy alaptelepítést kapjunk. Mindenfajta rendszermag-fordítgatás után sem kaptam rendes képet: hol kicsúszott, hol pedig mintha egy kifogyott szalagú mátrixnyomtatóból származó képet láttam volna. Rövid küzdelem után felhívtam egy ismerősömet, aki egy héten keresztül már küzdött hasonló nehézséggel. Két perc múlva az eredmény egy nem hivatalos SIS-rendszermagfoltra való hivatkozás volt. Hozzávetőlegesen tíz perccel ezután legalább már a konzolon olvasható, sőt kifejezetten szép betűkészlettel jelentek meg a feliratok. Ezekután gyerekjáték volt az X 4.2-es változatához egy hasonló foltot találni, aminek segítségével nagyjából tíz perc alatt tökéletes 1024×768-as képet sikerült elővarázsolni. Ahogyan a rendőrségi helyszínelők jegyzeteiben szokott szerepelni, „innentől kezdve az események felgyorsultak”.

Az `lspci` program segítségével megállapítottam, hogy egy Systems [SiS] SiS7012 PCI Audio Accelerator hangkártyával nézek farkasszemet. A 2.4.20-as

rendszermag ezt remekül támogatja. Azt lehet mondani, hogy mintegy két óra alatt egy olyan megbízhatóan működő rendszert tudtam felállítani, amiben csak a 56 K-s Winmodem nem működött – azóta viszont ezt is sikerült felélesztenem. Sajnos a nem hivatalos Winmodem-támogatást csak 2.4.18-as rendszermagra fordították le, és a körülbelül 1 MB-os forrás, amiből tetszőleges változathoz lehetne fordítani, csak 2 teljes nap alatt csurgott le az egyik gépemre, ami mellesleg a BIX hálózatán csücsül. De megérte, mert hozzávetőleg tíz perc alatt az is elkezdett muzsikálni. Nos, nézzük akkor részletesen!

A VGA konzol és az X beállítása

Egy Silicon Integrated Systems [SiS]: Unknown device 6325 nevű eszközzel kell felvennünk a küzdelmet, amihez a ➲ http://www.winischhofer.net/linuxsis630.shtml címen találunk konzolfoltot. A leírás rendkívül egyértelmű, de a foltozás után azért egy rendszermagfordítás szükséges hozzá. Az X foltozása is egy egyszerű `cp` parancs, ugyanis csak egyetlen SO-meghajtót kell bemásolni a megfelelő könyvtárba – ez a *readme.txt* állományban egyértelműen le van írva. Utána a `dpkg-reconfigure xserver-xfree8` parancs kiadásával hamar túleshetünk a dolgon. A végeredmény: borotvaéles X és gyönyörű konzolbetűk.

Hangkártya

A 2.4.20-as rendszermagban található „gyári” Intel ICH (i8xx), SiS 7012, nVidia nForce Audio or AMD 768/811x lehetőséggel tökéletes hangminőség érhető el.

Hálózati kártya

Ugyancsak a 2.4.20 rendszermagban a – SiS 900/7016 PCI Fast Ethernet Adapter support bekapcsolása után azonnal használhatjuk.

USB és IrDa

Az USB- és IrDa-támogatás beállítása egyszerű feladat, akinek azonban gondja lenne az alaplapi IrDával vagy a 2.0-s USB-vel, az a ➲ http://www.guska.hu címen megtalálhatja beállításuk módjait. Érdemes megemlíteni, hogy a gyári csomagban egy ajándék optikai egér bújik meg, amit Input Core eszközként fordítva tudunk használni – nagyon hasznos darab.

Szoftmodem

A ➲ http://www.linuxant.com/drivers/hcf/downloads-license.html oldalról a változatfüggetlen forráskódot érdemes letölteni, ami hozzáfordítja magát a pillanatnyi rendszermaghoz.
Hogy ne legyen minden tökéletes, az APM sajnos egyik rendszermaggal sem működik, ezért az `apmd` sem tudja irányítani a gépet. Az ACPI-támogatás viszont nagyon ígéretes. A rendszermagba belefordítva és az `acpid`-t feltelepítve legalább a *soft off* szolgáltatás és a belső akkumulátor figyelését el tudjuk érni. Apropó, az elem közel 3,5 órás üzemidőt nyújt számunkra, akár DVD-nézegetés közben is. Ha hajlandóak vagyunk játszadozni egy kicsit, és felrakjuk a legújabb fejlesztői 2.5-ös rendszermagot, akkor a benne lévő ACPI-támogatással egy

```
echo "xy MHZ" /proc/acpi/....."
```

parancs révén akár a processzor órajelét is meg tudjuk változtatni.

WLAN

Nem is olyan régen az FSF.hu aktivistái az LME jóvoltából kint jártak Hamburgban, az Openoffice.org első tanácskozásán. Abban a szerencsés helyzetben voltam, hogy magam is részt vettem a rendezvényen. Az eseményről az előző Linuxvilágban lehetett olvasni, ***Verók István*** tollából. Így most csak arra térnék ki, hogy milyen érzés volt ősembernek érezni magunkat olyanok társaságában, akik mind fejlett módszereket használnak. A konferenciaszervezők rendkívüli módon ügyeltek arra, hogy a világ minden részéről egybegyűlt „fontos emberek” gond nélkül hozzá tudjanak férni az Internethez. Így rendelkezésünkre állt több nagy teljesítményű Sun munkaállomás és WLAN-hozzáférés, természetesen UTP-kábeles csatlakozás is. Mivel szinte mindenkinek volt WLAN PCMCIA kártyája, nem fordítottak túl nagy figyelmet az UTP-s csatlakozókra. Mi is pórul jártunk, ugyanis vagy félóra keresgélésünkbe telt, mire találtunk egy jelölt csatlakozási pontot, ahol viszont 220 V nem volt. Nem beszélve arról, hogy szereplésük közben az előadók WLAN segítségével az Internetet a terem bármely pontjából használhatták, így ***Miguel de Icaza*** is bemutatott egy SmartMoney weboldalt. Nos, ott megfogadtam, hogy a legközelebbi konferenciára én is beszerzek egy WLAN-adaptert. Ezek ára az elmúlt időszakban elérhető lett, egy PCMCA-s változat ára 13–20 ezer forint között mozog. Az Asusnál árulnak olyan változatot az M2400E-ből, amiben van ilyen; bár az utólagos beépítési lehetőség ebben is adott, nem érdemes élni vele, ugyanis a többszörösét kóstálhatja egy PCMCIA-s modell árának. A nagykereskedésben kértem is egy Asus 11 Mbit/s PCMCIA-s modellt, aminek már a dobozára is rá van írva, hogy működik Linuxszal. A meghajtóprogramot a ➲ http://www.linux-wlan.com oldalról lehet hozzá megszerezni, a ➲ http://www.hup.hu oldalon pedig egy színvonalas beüzemelési útmutató található.

Összegzés

Az Asus M2400E rendkívül hasznos útitárs, amiben ugyan minden SIS alapú, de remekül működésre bírható. Érdemes viszont megemlíteni, hogy nagy terhelés esetén a gép szinte megvonja tőlünk a vezérlés lehetőségét. Például egy nagy sávszélességű letöltés esetén 2 MB/s-nál az egér akadozik, és a videofrissítés akár soronkénti is lehet. Hiába a Pentium 4-es teljesítmény, a sok eszköz (VGA, hangkártya) azonos IRQ-t használ, és falja a processzorunkat. Ugyanekkora asztali gép – például egy Intel Ether Express hálózati kártya – esetében 2 MB/s-os letöltésnél még egy Pentium III-as esetén is tudok teljes képernyős filmeket nézni, hiszen a hálózati kártya saját processzorral rendelkezik, így nem programból kell emulálni.

Varga S. Csaba
(guska@guska.hu)
Az 1.1-es Slackware óta linuxozik. Kedvtelései közé tartozik a fotózás és a Linux telepítése PDA-kra. Legszívesebben a Gerecsében túrázik a barátaival.

KAPCSOLÓDÓ CÍMEK

Linux-laptopoldal
➲ http://www.linux-laptop.net
SIS VGA meghajtó
➲ http://www.winischhofer.net/linuxsis630.shtml
HCF szoftmodemmeghajtó
➲ http://www.linuxant.com/drivers/hcf/downloads-license.html
Processzor-órajelállítás
➲ http://privat.uwe-schlenther.de/elinux_m2400.html
Wlan Driver
➲ http://www.linux -wlan.com
Asus nagykereskedés
➲ http://www.sowah.hu

Kell egy kis nosztalgia...

Te is unod a mai 3D-s csodákat? Vágyódsz a régi VGA-s grafika után? Ismét át szeretnéd élni a hőskori játékok hangulatát? Akkor a DOSBox a te barátod!

Talán akadnak még olyanok, akik emlékeznek a régi szép időkre, amikor egy álmatlan éjszakán bekapcsoltuk a 386-osunkat, ami egyből buzgón meg is számolta mind a 8 MB memóriánkat, majd szépen betöltötte az MS-DOS-t. Izgatottan vártuk, hogy megjelenjen a bűvös felirat, és elindíthassunk egyet a ma már legendának számító játékok közül: az Alone in the Darkot vagy a Dune2-t.
Ezek az idők már elmúltak, most már több gigaherzes processzorok dübörögnek a gépünkben, memóriánk az akkori merevlemezünk méretének többszörösére duzzadt és valahogy már DOS-t sem használ senki.
Jó hír azok számára, akik nem találják a helyüket a 3D-s játékok világában, vagy csak nosztalgiázni szeretnének egy kicsit, hogy nem kell kétségbeesniük, mert megszületett a DOSBox nevű megoldás, amivel könnyen és egyszerűen futathatjuk régi DOS-os játékainkat.
A DOSBox egy DOS-emulátor, amit kifejezetten régi játékok futtatására fejlesztettek ki. A 286-os, illetve a 386-os valós módját emulálja, ez azt jelenti, hogy azok a játékok, amik védett módot használnak (például Duke Nukem 3D), sajnos nem futtathatók alatta.
Azok a játékok azonban, amik nem használnak védett módot, szinte kivétel nélkül futnak. A DOSBox támogatja az XMS és EMS memóriakezelést, továbbá képes emulálni a Sound Blaster hangkártyát is. Így nem kell lemondanunk a hangokról sem. Ezenkívül képes egy általunk megadott könyvtárat DOS-os fájlrendszerként emulálni (lásd később).
A DOSBox egy SDL (Simple DirectMedia Layer) nevű könyvtárcsomagra épül, ami felületfüggetlen multimédiás szolgáltatást nyújt. E szolgáltatásokba beletartozik például a billentyűzet, az egér és a botkormánykezelés, a hangra és az OpenGL-re épülő 3D-s megjelenítés. Az SDL most már rengeteg felület alá elérhető, például Windowsra, BeOS-re, a BSD-kre és a Solarisra is. (Az SDL a ➲ http://www.libsdl.org oldalról tölthető le, de a Linux-terjesztések többségében is megtalálható).
Az SDL-lel készített alkalmazások így percek alatt átvihetők egyik felületről a másikra. Ezért a dosbox sem kizárólag Linux alatt fut, hanem windowsos és BeOS-es változata is létezik.
A másik fontos dolog, amit nem árt tudnunk a DOSBoxról, hogy folyamatosan fejlesztés alatt áll. A jelenlegi változat (a 0.58-as) már tartalmazza a legfontosabb szolgáltatásokat, és a játékok jelentős hányada elindul alatta. A tapasztalatok szerint azonban még korántsem képes tökéletesen emulálni a DOS és a BIOS szolgáltatásait, és jelentősebb hiányosságok is akadnak benne, például az SVGA és a VESA grafika használatára nincs lehetőség. A fejlesztők azonban szorgalmasan dolgoznak, folyamatosan bővítik az alkalmazást, így egyre jobban növekszik a futtatható játékok köre.
Mindezek ellenére már most is rengeteg játékot támogat. A DOSBox honlapján, a ➲ http://dosbox.sourceforge.net/comp_list.php?orderby_name&letter= címen egy teljes listát találhatunk arról, hogy milyen játékokat futtathatunk rajta.

A telepítés

Az SDL-en kívül a DOSBox futtatásához a curses, a zlib és a libpng csomagokra is szükségünk lesz. Ezek is megtalálhatók bármelyik terjesztésben.
A DOSBox része a SuSE 8.1-es változatának, tehát ha van otthon 8.1-ünk, akkor megtalálhatjuk a telepítő CD-n. A legfrissebb változatot letölthetjük a ➲ http://dosbox.sourceforge.net/ címről. Itt megtalálhatjuk egyrészt a windowsos és a BeOS-es lefordított változatot. Binárisban letölthető RPM-ben is, ám ez nem minden terjesztés alatt fut, ezért azt javasoljuk, hogy inkább a forrást szedjük le, és fordítsuk le mi magunk.
A fordításhoz először csomagoljuk ki a `tar -xvzf dosbox-0.58.tar.gz` parancs segítségével, majd lépjünk be a létrejött könyvtárba. Először adjuk ki a `./configure` utasítást, ami ellenőrzi, hogy megvan-e minden szükséges összetevő. Ha nem kaptunk hibaüzenetet, a `make` paranccsal kezdhetjük a fordítást. Ezután a `make install` utasítással telepíthetjük a lefordított összetevőket, de ne feledjük, hogy ezt csak rendszergazdaként tehetjük meg.

Ha ez is megvan, akkor keressük elő régi lemezeinket a kedvenc hőskori játékainkkal. Hozzunk létre egy könyvtárat, és másoljuk oda őket. Ezután indítsuk el a DOSBoxot. Egy új ablak fog megjelenni, amiben egy `dos` héj fut. Írjuk be a `mount c [k nyvtÆr]` parancsot. A könyvtár itt azt a könyvtárat jelöli, ahová a játékainkat másoltuk, és úgy kell megadnunk, ahogy a Linux alatt is hivatkoznánk rá, például */home/user/dosgames*. Ezután a `C:` lemezrészen keresztül érhetjük el ennek a könyvtárnak a tartalmát, és máris indíthatjuk kedvenc játékunkat.

Honnan szedjünk régi játékokat?

Mit kell tenni akkor, ha régen nem gondoskodtunk játékaink mentéséről és tárolásáról? Szerencsére az Interneten sok régi, feltételekhez kötötten ingyenes (shareware), ingyenes (freeware), illetve azóta már szabaddá tett DOS-os játék érhető el. Érdemes körülnéznünk a ➲ http:/www./dosgames.com címen, illetve a ➲ http:/www.hanbyl.com/ClassicGames oldalon. Ezek valamelyikén biztosan fellelhetjük egyik-másik régi kedvencünket.

A cikkhez kapcsolódó anyagok a 48. CD Jatek könyvtárában találhatóak.

Garzó András (garzoand@interware.hu)
Körülbelül három éve foglalkozik Linux- és más Unix-rendszerekkel. Legjobban az operációs rendszerek lelkivilága érdekli, de nyitott egyéniség.

Játsszunk, játsszunk valamit...!

Ismét itt a nyár, süt a nap és jön a forróság. Mehetünk strandra, nyaralni, de éppenséggel maradhatunk itthon is, a szoba hűvösében.

Amennyiben az utóbbi mellett döntünk, akkor sem fogunk unatkozni, hiszen rengeteg új játék jelent meg kedvenc rendszerünkhöz.

Foobillard

Ez egy igazi OpenGL-re épülő 3D-s „biliárdszimulátor”, elsősorban megszállottaknak. A játék büszkén hirdeti magáról, hogy a fizikai törvényeket egy az egyben betartja, és ezáltal valósághű élményt nyújt. Valóban, a golyók tényleg arra szaladnak, amerre lökjük őket, de a szemfülesebb játékosok észrevehetnek néhány benne felejtett hibát. Sebaj, talán majd a következő változatra kijavítják. Mindenesetre ezek a hibák nem olyan súlyosak, hogy a játékélményt komolyan befolyásolnák.

Azt viszont meg kell hagyni, hogy a játék nagyon szép. A golyók kidolgozottak, sőt még árnyékuk is van. Az asztalt szabadon forgathatjuk bármelyik irányba. Az már csak hab a tortán, hogy mi választhatjuk meg az asztalt fedő posztó színét is.

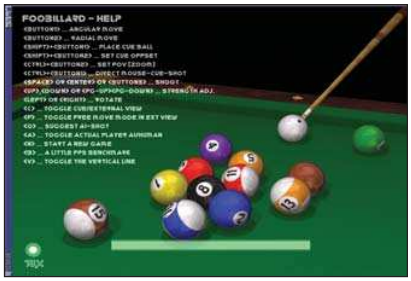
Mint minden rendes biliárdprogram, ez is többféle játékot tesz lehetővé. A hagyományos pool mellett játszhatunk 9 ballt és snookert is. Lehetőség nyílik a többjátékos módra mind helyi, mind hálózati szinten. Emellett a géppel is összemérhetjük a tudásunkat, de csak ha nem félünk a megaláztatástól.

A Foobillard mind forrásban, mind RPM-ben letölthető. Két változata is van: az egyik az SDL-re (Simple Direct-Media Layer), a másik a GLUT (openGL Utility Toolkit) könyvtárcsomagra épül. A Linux-terjesztésekben általában mindkettő megtalálható. A különbség a két változat között lényegében annyi, hogyha az SDL-eset töltjük le, akkor a játéknak hangja is lesz.

➲ http://sunsite/foobilliard.dk

Barrage

Az xbill után (amikor a Windows logóval mászkáló Billeket kell szétütni – alapmű) itt a következő őrület, ami garantáltan álmatlan éjszakákat és „kattintóizomlázat” okoz. A Barrage-ban egy nehéztüzérségi ágyút „alakítunk”, és a feladatunk nem



kevesebb, mint elpusztítani mindenkit, aki átmegy a területünkön.

A képernyőn három különböző típusú egység mászkál, fentről lefelé, illetve lentről fölfelé: gyalogosok, jeepek és tankok. Minden elpusztított gyalogosért 5 pont, jeepért 20 pont, tankért pedig 50 pont jár. Igen ám, de minden olyan jeepért, ami sértetlenül halad át a területünkön, 10 pontot kell fizetnünk. Tankoknál az ár 50 pont. Szerencsére a gyalogosoknál ilyen nincs.

Kétféle fegyverrel tüzelhetünk: az első egy golyószórószerű fegyver, amit a középső egérgombbal tudunk használni. Ez hatékonyan vethető be a gyalogosok és a jeepek ellen, mivel gyorsan egymás után tudunk tüzelni, viszont a tankok ellen hatástalan. A tankokat csak a bal egérgombbal kilőhető löveg tudja elpusztítani, ezt viszont csak három másodpercenként lőhetjük ki.

A játék célja, hogy három perc alatt minél több pontot gyűjtsünk össze. Ez azonban komoly kihívás, e cikk írójának például egyórányi játék alatt sem sikerült három számjegyű pontszámot elérnie. Mindezek tetejébe a Barrage nemcsak nehéz, hanem komoly függőség kialakítására is képes.

➲ http://lgames.sourceforge.net/index.php?project=Barrage

Lemmingek után szabadon

Ki ne emlékezne a Lemmings című játékra, ami új fejezetet nyitott a logikai játékok világában? A lemmingek ideje azonban lejárt, helyükre a pingvinek léptek. A Pingus egy az egyben ugyanaz, mint a nagy elől, csak – értelemszerűen – pingvinekkel. Természetesen a pingvinek sem képviselnek magasabb szellemi szintet – szegények értelmileg egy picit visszamaradottak, maguktól semmi másra nem képesek, mint menni az orruk (pontosabban a csőrük) után.

Azt viszont nagyon bátran teszik, ha nem avatkozunk közbe, simán belesétálnak a legmélyebb szakadékba is.

A feladat továbbra is ugyanaz: átvezetni szegényeket a pályán olyan módon, hogy a lehető legkevesebb pingvin essen áldozatul a zord körülményeknek.

Néhány bevezető tanulószint után egyre durvább pályakkal kell szembesülnünk, amik olykor-olykor komolyabban is megerőltethetik szürkeállományunkat.

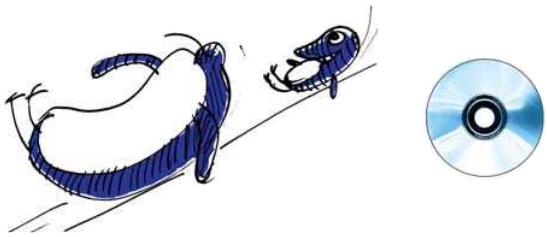
A grafikára igazán nem lehet panaszunk, nagyon aranyosan gyalogolnak pártfogoltjaink, a zene is sokat dob a hangulaton. Az egyetlen negatívum, hogy még mindig viszonylag sok hiba maradt benne, de ezek az újabb változatok során remélhetőleg el fognak tűnni. Mindenesetre ez a játék is remek munka utáni, illetve előtti (esetleg helyetti) elfoglaltságot jelent:

➲ http://pingus.seul.org.

A cikkhez kapcsolódó anyagok a 48. CD Jatek könyvtárban találhatóak.

Garzó András (garzoand@interware.hu)

Játszótér



Quake 1, 2, 3 és...

Mint az a címből is kitűnik, e havi témánk a Quake-sorozat – a kezdetektől egészen napjainkig. Az id Software honlapján a teljes termékkínálat megtalálható, természetesen nemcsak a Quake-sorozat, hanem minden id-fejlesztés, régiek-újak együttesen. Ennek ellenére most maradjunk a Quake bűvöletében.

Quake 1

Lássuk a már történelemmé nemesült kezdeteket! A program 1996-os megjelenése minden addigi elképzelést felülmúlt: a DOS operációs rendszeren nevelkedett játékosok addig nem létező képi világgal szembesülhettek.
John Carmack és társai olyan játékot adtak ki a kezük közül, aminek az OpenGL és az általa nyújtott leképezési rendszer volt az alapja. 1997-ben jelent meg a glquake, valamint az első Unix-, illetve Linux-változatok. A glquake windowsos és linuxos változatai 3Dfx-re, és az általa támogatott Glide-ra készültek. Figyelembe véve az eltelt időt és a Glide, illetve a 3Dfx kártyák kiszorulását a gépeinkből, a glquake telepítésével nem foglalkozunk (ennek ellenére a 3Dfx kártyatulajdonosok számára a CD-mellékletre felraktuk a glquake-0.97 változatát, valamint – ha szükséges – elektronikus levélben kész vagyok telepítési útmutatást adni).

Telepítés

A squake-1.1 (SVGA-változat) futtatásához szükséges elemek

- SVGALib 1.20 vagy újabb (/lib/libvga.so.1.2.10) változata;
- libc 5.2.18 vagy frissebb (5.0.9-vel nem működik, /lib/libc.so.5.2.18) változata;
- CD-ROM a CDAudiónak;
- SoundBlaster 16 és Gravis Ultrasound MAX hangkártya;
- SVGALib által támogatott egér (az egér általában működik az squake alatt, amennyiben X alatt is használható);
- 2.0.24 vagy újabb rendszermag.

Útmutató lépésről lépésre

1. Nyiss egy `xterm` ablakot!
2. Add ki a `su root` utasítást, a rendszergazda jelszavával lépj be a rendszerbe.
3. A `cd` paranccsal lépj a */usr/local/games/* könyvtárba.
4. Az `mkdir quake` paranccsal hozz létre egy ***quake*** könyvtárat.
5. Fűzd be a Quake CD-t, és a CD gyökeréből másold át a ***q101_int.1*** és ***q101_int.2*** fájlokat a ***quake*** könyvtárba.
6. A `cat q101_int.1 q101_int.2 > quake.1` paranccsal egyesítsd a fájlokat.
7. A `lha e quake.1` paranccsal csomagold ki a teljes ***quake.1*** archív állományt, vagy a szintén a CD-mellékelten lévő *../pack/* könyvtárban lévő ***id1.tgz*** fájlt másold a ***quake*** könyvtárba.
8. A `tar xvfz- fÆjlneve` utasítással csomagold ki (ekkor jön létre az ***id1*** könyvtár).
9. A CD-mellékleten lévő *../quake* könyvtárban lévő ***squake-1.1.tar.gz*** fájlt másold a quake könyvtárba.
10. A `tar xvfz- fÆjlneve` utasítással csomagold ki.
11. Majd a `./squake` paranccsal futtasd az squake programot.

Természetesen nem ilyen egyszerű a feladat. Figyelembe kell venni, hogy az squake elkészítése óta nagyon sok idő telt el, és a program együttműködési hajlama valószínűleg a nullához közelít, nincs hozzá terméktámogatás sem, mivel a fájl az ➲ ftp://ftp.idsoftware.com tárhelyen, az ***unsup*** könyvtárban található. Fontos az is, hogy a Quake 1 forrását az id Software kiadta és a GPL felhasználási szerződés alatt szabadon felhasználhatóvá tette. Írásunknak viszont nem célja bármilyen, a forrás felhasználásával készített Quake-változat telepítésének támogatása és leírása.
A telepítés során több gond is felmerülhet, ezek közül az első, hogy az egyes Linux-kiadásokban a tömörítő programok nem mindegyike kerül fel a gépre, vagyis az `lha` nem feltétlenül képezi az alaptelepítés részét, ezért ezt külön fel kell tenni.
A következő nehézséget a libc5-tel való megfelelőség okozhatja. A SuSE 8.2 alatt az ***shlibs5.rpm*** csomagot kell telepíteni, ez gyógyír a bajra. Red Hat 9 alatt sajnos nem találtam rá az együttműködő RPM csomagra, így elővettem a régi 6.2-es lemezét, és `rpm -ihv` paranccsal felte-

lepítettem a ***libc***, valamint a ***ld.so*** csomagot (a megfelelő változatok szintén fent vannak a CD-mellékleten). Ezen túljutva az ***svgalib*** okozta galibába ütközünk. Red Hat esetében ismét a 6.2-es CD-jéhez kell nyúlni, míg SuSE-nél az ➲ ftp://ftp.suse.com tárhelyről kell feltelepítenünk a megfelelő változatú ***svgalib***-et. A sokak által használt Mandrake esetében – mivel akkortájt még nagymértékben hasonlított a Red Hatre – ugyanúgy kell eljárni (minden szükséges RPM-et, kivéve a SuSE ***shlibs5.rpm*** csomagot, tartalmazza korongunk). Fontos, hogy az svgalib, valamint az squake közvetlen módon akarja írni, illetve használni a videokártyát, ezért a következő parancsokat rendszergazdai jogosultsággal kell kiadnunk:

```
chown root squake
chmod 4755 squake
```

A quake.x11-1.0 (X11 változat) futtatásához szükséges elemek

- X11R5 vagy újabb változat, a program XFree86 alatt lett kipróbálva, de általában egyéb X-kiszolgálókkal is működik;
- libc 5.2.18 vagy újabb változat (5.0.9-cel nem működik, ezért használjuk a */lib/libc.so.5.2.18*-at);
- CD-ROM CDAudiónak;
- SoundBlaster 16 és Gravis Ultrasound MAX hangkártya;
- SVGALib által támogatott egér (az egér általában működik az squake alatt, ha X alatt is működik);
- 2.0.24 vagy újabb rendszermag.

A telepítési lépések megegyeznek az squake esetében leírtakkal, eltérés csak abban van, hogy a CD-mellékletről a ***quake.x11-1.0.tar.gz*** állományra szükségünk van, és a telepítést követően a `./quake.x11` parancsot ki kell adni. Új elem továbbá a X11R5-tel való megfelelőség elérése: SuSE esetén a ***shlibs5.rpm*** felrakásával megoldódik, míg Red Hat 9 alatt a program gond nélkül futott X11R6-tal is.

Quake 2

1997 végén megjelent a Quake 2, továbbfejlesztett 3D-támogatással, ami még mindig erőteljesen a 3Dfx tulajdonosoknak kedvezett, bár már az első

Május hónapban a linuxos játékhírek sokaságát ontotta magából az Internet. Íme az általam legizgalmasabbnak ítéltek!

Icculus-hírek

Számos linuxos játék fordítását végzik itt. A cég bejelentette, hogy a Devastation Linux Server próba1-es változatot elérhetővé és letölthetővé tették. A bejelentés a ➲ http://icculus.org/news/news.php?id=1416 címen olvasható, míg az állomány a ➲ http://ftp.eongames.com/iucculus.org/devastation/devastation-lnxded-380.tar.bz2 helyről tölthető le. A Devastation játékoldala a ➲ http://www.devastationgame.com/ címen érhető el.

Az Uplink fejlesztői korongon

Az Uplink készítői egy úgynevezett developer CD-n kiadják a program forrását! Mivel azonban a program kereskedelmi termék, a forrást feltehetőleg nem ingyen teszik közzé.
➲ http://introversion.co.uk/uplink/
➲ http://introversion.co.uk/uplink/news.html

Megjött a WineX 3.0 Pont2Play!

Megjelent a Transgaming által fejlesztett WineX 3.0 Pont2Play elnevezésű változata, amivel egyre több és jobb windowsos játék futtatható Linux alól.
Egy elkövetkező számban részletes leírást fogok közölni a programról, valamint a beállításáról, ehhez viszont még meg kell nyernem a Transgaminget, hogy a WineX-et feltehessük a magazin CD-mellékletére.
➲ http://www.transgaming.com
➲ http://www.transgaming.com/download.php

Új játék a láthatáron!

A program neve StokedRider, és az év vége felé várható a megjelenése. A program egy

extrém snowboardjáték, az előzetes képekből az már most látszik, hogy nem csak havon lehet majd száguldozni.
➲ http://www.stokedrider.com/index.htm

Enemy Territory próba

A hónap valódi érdekessége, hogy megjelent az Enemy Territory első próbaváltozata, ezt követően a készítők „feltehetőleg" a teljes játékanyagot kiadják. Más játékok küldetésanyagaival szemben itt nincs szükség az eredeti RTCW (Return to Castle Wolfenstein) lemezre, illetve telepítésre.
➲ http://gamesactivision.com/games/wolfenstein

Új NWN beta4

Ismét megjelent a NWN (Neverwinter Nights) újabb próbaváltozata, 4-es számmal.
➲ http://www.nwnbioware.com/downloads/linuxclient.html

Midnight Racing Linuxra

A Hyperion Entertainment újabb linuxos átültetésbe kezd: az eddigi kínálatot bővítve

(SiN, Shogo MAD) a Midnight Racing autóverseny program átültetését tervezik Linuxra.
➲ http://www.hyperion-entertainment.com/
➲ http://www.incagold.com/products/midnight20racing/midnight.php

nVidia TNT kártyákat is támogatta, de a teljesítménypróbákon az akkoriban kialakult versenyben továbbra is a 3Dfx volt a nyerő.
Tavaly, amikor a program elérte a 3.20-as

változatot, követte a Quake 1-es sorsát, és a forrását nyílttá tették. Nem elhanyagolható tény, hogy ennek a programnak a grafikai motorjára rengeteg másik játékot írtak, ilyen például a SiN, Wage Of SiN, Half Life-sorozat, Soldier Of Fortune; sőt ebbe a motorba belepiszkálva, illetve feljavítva hozták létre a Daikatana programot.

Telepítés

A quake2-3.20-glibc….tr.gz futtatásához szükséges elemek

- SVGALib konzolgrafika (*ref_soft.so*) szükséges, az SVGALib 1.2.0 vagy újabb változat;
- X11 ablakos grafika (*ref_softx.so*), X11R5 vagy újabb változat;
- 3DFX fxMesa, Mesa 3D vagy 3DFX Miniport (*ref_gl.so*), Mesa 3D 2.6 vagy újabb, kifejezetten 3DFX támogatással fordítva;
- általános glX (X11) OpenGL alapú (*ref_glx.so*).

Lépések…

1. Rendszergazdaként lépj be a */usr/local/games* könyvtárba.
2. Az `mkdir quake2` paranccsal készíts egy *quake2* könyvtárat.
3. A fent említett fájlt másold át a CD-mellékletről.
4. Add ki `tar xvfz- fÆjl neve` utasítást.
5. A libGl.so, libMesaGL.so, libMesaGL.so.2 hivatkozásokat és a libMesaGL.so.2.6 állományt törölni kell.
6. A *quake2.conf* fájl át kell helyezni a */etc*-be.
7. A windowsos telepítő CD-ről, az *Install/Data/baseq2* könyvtárból a *pak0.pak* fájlt be kell másolni a */usr/local/games/quake2/baseq2* könyvtárba.
8. A *baseq2* tartalmaz egy *players* könyvtárat is, ebben bújik meg a *female* és a *male* könyvtár, ezekben a könyvárakban lévő *.pcx* kiterjesztésű fájlokat be kell másolni a */usr/local/games/quake2/baseq2/players/female*, illetve */usr/local/games/quake2/baseq2/players/male* könyvtárba.

Kész is, így viszont még nem mennek az összekötő és bevezető (intro) videók. Ezt úgy érdemes megoldani, hogy a telepítő CD *...baseq2/video* könyvtárát a `ln -s` parancs segítségével teljes egészében linkelni kell a */usr/local/games/quake2/baseq2* könyvtárba. Egyetlen dolog hiányzik még: egy közvetett hivatkozás az OpenGL vezérlőre, a */us/lib/libGL.so*-ra. Érdemes erre a fájlra mutatva készíteni a közvetett hivatkozást, mert így a meghajtó vezérlőjének váltásakor nem kell minden alkalommal elkészíteni a közvetett hivatkozást.

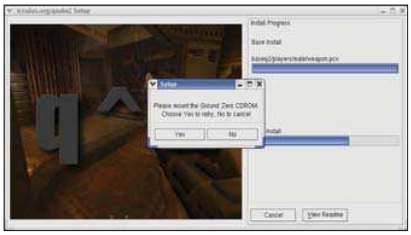
```
ln -s /usr/lib/libGL.so /usr/
➥local/games/quake2/libGL.so
```

A Readme fájl szerint a *quake2* fájlnak továbbra is szüksége van a videoalrendszer-

hez történő egyedi hozzáférésére, tehát ki kell adnunk a következő parancsot:

```
chown root quake2
chmod 4755 quake2
```

Innentől kezdve nincs másra szükség a játék indításához, mint futtatni a `./quake2` parancsot – ehhez viszont egyedi paraméterek kellenek.



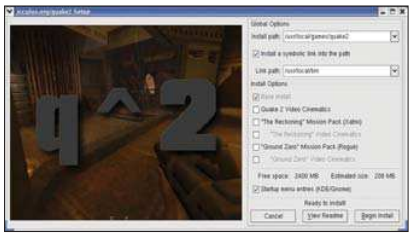
Így érdemes futtatni a programot:

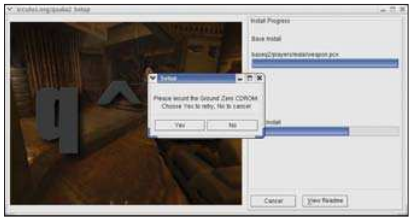
```
./quake2 +set vid_ref glx +set
➥gl_diver libGL.so
```

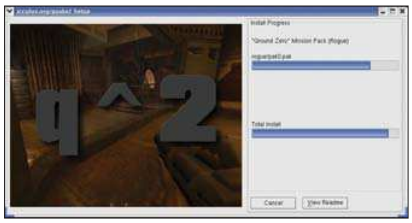
(A fenti példából adódik, hogy a ***libGL.so*** a ***/usr/local/games/quake2*** könyvtárban található, és a megfelelő vezérlőre mutat).
Mindezt egy egyszerű parancsfájlba is be lehet foglalni:

```
#!/bin/sh
##############################
# Load Quake2 from
# /usr/local/games/quake2
##############################
cd /usr/local/games/quake2
➥./quake2 +set vid_ref glx
➥+set gl_driver libGL.so
```

(A ***q2load***, valamint a ***quake2.ico*** fájl CD-mellékletünkön ugyancsak megtalálható.)
A játék csak ablakban működik 3D-gyorsítással, és X alatt nem kezeli az egeret, de ennek a nehézségnek létezik megoldása. Méghozzá két választásunk is adódik: a ➲ http://icculus.org oldalon rengeteg windowsos programot próbálnak meg Linuxra ültetni, az egyik ilyen projekt keretében készítettek egy grafikus telepítőt a Quake2-höz (➲ http://icculus.org/quake2). A fájlt ***i_o_quake2-0.15.run*** néven kell keresni (lásd CD-nken), ez futás közben bekéri az eredeti Quake2 windowos korongot, valamint kezelni képes a Quake2-höz készült küldetéslemezeket is.







A telepítőprogram a q2hack bővítményt is tartalmazza (ez külön is szerepel a CD-nken), ami nem más, mint egy DGA-bővítmény a Quake2 programhoz, hogy az képes legyen teljes képernyős üzemmódban futni, valamint X11 alatt kezeli az egeret. Fontos megjegyezni, hogy a program a Quake2 GPL forrására épül, és nem azonos az id Software által kiadott linuxos anyaggal.
Amennyiben az eredeti id Software-es anyagot meg akarjuk tartani, de szükségünk lenne a teljes képernyős üzemmódra, akkor a ***q2hack*** programot kell telepítenünk. A ***q2hack***-nek az id Software termékre való telepítése roppant egyszerű:

```
tar xvfz- q2hack.tar.gz
```

majd rendszergazdai jogosultsággal adjuk ki a szokásos utasításokat:

```
make
make install
```

Így megváltozik a futtatandó állomány. Nem a Quake2 programot kell meghívni a játékhoz, hanem a ***q2*** parancsfájlt, ami már a DGA-támogatással futtatja a Quake2 programot.

Quake3

John Carmack tovább növelte az OpenGL-ből kihozható energiát, majd új, korszakalkotó grafikai motort készítve létrehozta a Quake3-at. Olyan programot alkotott, ami jó testreszabhatóságának köszönhetően mindenféle grafikus kártyán futni képes. 1999 végén megjelent a Quake3 linuxos változata. Körülbelül itt kapcsolódik be a linuxos játékok történetébe a Loki. A cég által rengeteg windowsos játék került fordításra, illetve átültetésre Linux alá. Több oka is van annak – és ezeket nem tisztem vitatni és vizsgálni –, hogy a Loki megszűnt, ám az általa végzett munka máig fennmaradt (lásd az nVidia új telepítőprogramját).

Felpakolás

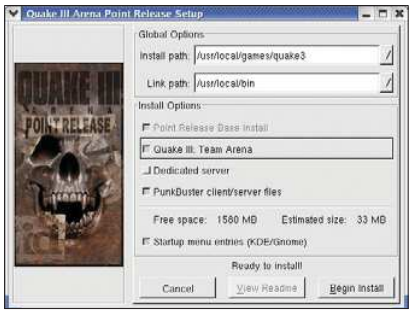
A játék telepítése kétféleképpen is történhet: az eredeti linuxos CD-ről való telepítéssel, valamint a linuxos frissítési, illetve kiadási pont meglátogatásával, ami tartalmazza a szükséges linuxos bináris állományokat, valamint a frissített ***pak*** kiterjesztésű fájlokat, amiket letölthetünk.

Dióhéjban a DGA-ról

A DGA (Direct Graphics Architecture) közvetlen grafikus felépítést jelent.
A DGA nem gyorsítás, csupán az X egyfajta kiegészítése, bővítése, ami nagyon csekély gyorsítást tartalmaz. Alapvetően a 2D-s leképezésben nyújt segítséget az XFree86 számára, valamint a programok számára a rendszer egér- és billentyűzet-jeleinek olvasásához közvetlen hozzáférést nyújt. Gyakorlatilag a felhasználó nem is látja, hogy egy-egy alkalmazása DGA-t használ, mivel azt az XFree86 tartalmazza, és a program, ha megfelelő jogosultsággal fut, hozzáfér a DGA-hoz.
➲ http://www.xfree86.org

Az első változat esetén be kell fűzni a CD-t, majd rendszergazdai jogosultsággal futtatni kell a `setup.sh` parancsállományt.

A *fenti képről* látszik, hogy a linuxos telepítőprogram sokkal több elemet telepít fel, mint a frissítés. Érdemes még megjegyezni, hogy a telepítőprogram felrakja ugyan a ***pak0.pk3*** fájlt, de azt utólag törölve ugyancsak tárhelyet takaríthatunk meg, ha a linuxos CD-n megtalálható */baseq3* könyvtárban lévő ***pak0.pk3*** fájlra készítjük el a közvetett hivatkozást, hasonlóan a lentebb leírt windowsos korongról való telepítés módszeréhez.
A második módszer esetében fel kell telepíteni a frissítést, majd be kell fűzni egy windowsos Quake3 CD-t.



Erről a szükséges fájl a ***pak0.pk3***, ezt kell bemásolni (megfelelő méretű merevlemez esetén), vagy az `ln -s` parancs segítségével közvetett hivatkozást kell róla készíteni a */usr/local/games/quake3/baseq3* könyvtárba a

```
ln -s /mtn/cdrom/Quake3/baseq3/
➥pak0.pk3
➥/usr/local/games/quake3/
➥baseq3/pak0.pk3
```

parancs segítségével.

A frissítőcsomag feltelepítése a linuxos CD esetén is szükséges, ilyenkor azonban csak a frissített ***pak*** fájlokkal kell foglalkozni, illetve alapértelmezésben a frissítés csak ezeket teszi fel. Nincs szükség a windowsos CD-re, mert ha a program alapvetően nem arról lett telepítve, akkor csak fölöslegesen nyúlnánk bele a telepített programkörnyezetbe. Erre többek között azért is szükség van, mert a kiegészítő lemezt, a Quake3 Team Arena programot csak a frissített pakfájlok segítségével lehet futtatni.

Quake 3 Team Arena

A Team Arena azokat a Quake3-rajongókat próbálja meg maga mellé állítani,

akik nem voltak megelégedve a Quake3 hálózatos játék jellemzőivel, valamint ezzel a küldetéslemezzel próbálták meg kárpótolni a felhasználókat az általuk hiányolt részekért.
A telepítés történhet eredeti linuxos CD-ről, de a felrakott frissítést követően windowsos korong segítségével is.
A windowsos út esetén program futtatásához szükség lesz az eredeti Q3 CD-re, mert ha eddig bárki csak közvetett hivatkozást használt (lásd a ***pak0.pk3*** esetében), akkor a windowsos lemezről történő telepítés esetén a 850 MB-os ***pak0.pk3*** fájlt kénytelen lesz felmásolni a */usr/local/games/quake3/baseq3* könyvtárba. Ez az alapja a kettős Quake3 és Quake3 Team Arena futtatásnak.
Fel kell telepíteni a fent említett frissítést, majd kölcsön kell kérni vagy meg kell venni egy Quake 3 Team Arena windowsos CD-t, és a következőket kell

tennünk: a beszerzett lemezről, a ***missionpack*** könyvtárból a ***pak0.pk3*** fájlt be kell másolni a */usr/local/games/quake3/missionpack* könyvtárba. A többi packfájlra azért nincs szükség, mert az id Software által kiadott frissítés tartalmazza őket.
Az indítás kétféleképpen is történhet: a régi Quake 3 programból, vagy az alábbi utasítássorral:

```
./quake3 +set fs_cdpath
➥"/usr/local/games/quake3/
➥baseq3" +set fs_game
➥missionpack
```

Ehhez érdemes egy rövid kis parancsállományt írni:

```
#!/bin/sh
##############################
# Load Quake3 Team Arena from
# /usr/local/games/quake3
##############################
cd /usr/local/games/quake3
➥./quake3 +set fs_cdpath
➥"/usr/local/games/quake3/
➥baseq3" +set fs_game
➥missionpack
```

CD-nken a mintafájl ***arena*** néven lelhető fel. Azzal, hogy az összes állomány felmásolásra, illetve telepítésre került, oda-vissza válthatsz a két program, még inkább a két küldetés között. Az eredeti

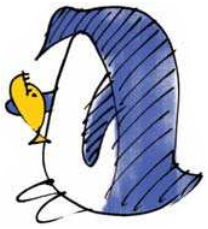
Quake3 az Arena alól is indítható. Kimondhatjuk, hogy az id Software a maga nemében páratlan játékot alkotott, és egy fantasztikus küldetéssorral ajándékozott meg bennünket. Mindhárom rész áttörés volt a maga idejében az addigi megjelenítés és a 3D-s grafika terén. Az id Softwaretől sokan liszenszelték a grafikai motort, amire számos játékot építettek. A jövő hónapban ezekkel foglalkozom majd, körbejárva a kínálatot, hogy – némi átdolgozással – más programozók, illetve cégek mit hoztak ki a Quake-motorokból.
(Megjegyzés: mellékletünkön a ***quake2***, valamint a ***teamarena*** könyvtárban találhatók meg a pályabővítmények és a cheatek is.)

Kosztandinovszki Norbert
(kosztandinovszk@dialec.hu)
Linux- és játékmániás számítógépőrült.

KAPCSOLÓDÓ CÍM

➲ http://www.idsoftware.com

Hasznos apróságok



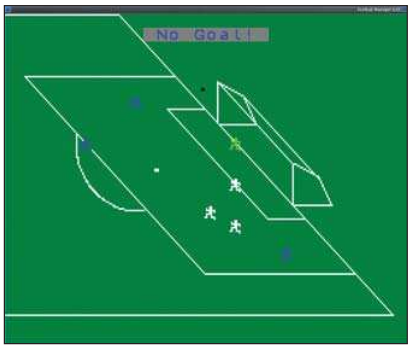
GNU Pilot Logbook Pro

A pilóták tudják, hogy a naplóvezetés maga nem jelent nagy munkát. A helyzet azonban bonyolódik, ha azt akarod tudni, hogy melyik óratípusból mennyivel rendelkezel. Ez a naplóprogram teljesen olyan, mint a hivatásos repülési naplók; minden szükséges bejegyzés megtalálható benne, illetve két, felhasználó által megadott mezőt is kínál. Ezenkívül egy kis héjprogramot – ezt saját magadnak kell megírnod – lefuttatva az adatfájlon egy olyan fájlt állíthatsz elő, ami 60 vagy 90 napra visszamenőleg összesíti az órákat, és kiszámítja, hogy mennyit jelentenek pénzben kifejezve. Előfeltételei: libgtk, libgdk, libgmodule, libglib, libdl, libXi, libXext, libX11, libm, glibc, pilótaengedély és repülőgép (e két utóbbi nem kötelező).

➲ ftp.stampede.org/skibum/gplbp/gplbp.tar.gz

Football Manager

Ebben a játékban egy futballcsapat edzője vagy. A grafika elnagyolt, de a játék igen szórakoztató. Stratégiai játékról van szó: játékosokat vásárolhatsz és adhatsz el, és ki kell választanod, hogy kik játsszanak a meccsen az adott héten.



Ha ezzel végeztél, szusszanj egyet, majd 30 másodperc múlva megnézheted a kapuralövéseket és az eredményt. Ezután megtekintheted a csapatod helyezését, vagyis hogy a liga többi csapatához képest előbbre vagy hátrébb kerültél. Ha nem törlöm le ezt a játékot a gépemről, soha nem végzek a munkámmal – még az Adventure-nél is csábítóbb. Előfeltételei: libSDL, libm, libX11, libXext, libdl, libpthread, glibc.

➲ http://www.autismuk.freeserve.co.uk

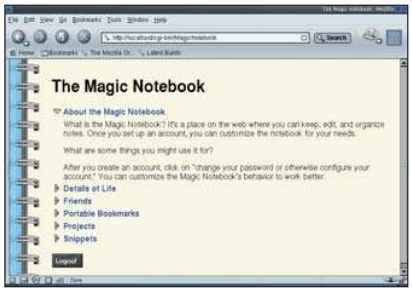
Hardware Lister

Ez az alkatrészlista-készítő ügyel a részletekre, és tartalmazza az IRQ-t, a használt modulokat, illetve további adatokat a kártyákról és más alkatrészekről. Ha például leltárkészítés miatt igen részletes adatokra van szükséged a rendszerről, akkor a figyelmedbe ajánlom ezt a programot. Egyedül a hálózati kártyák MAC-címét nem tartalmazza, az azonban elég egyszerűen kideríthető. Futtatásához libstdc++, libm, libgcc_s és glibc szükséges.

➲ http://ezix.sourceforge.net/software/lshw.html

Magic Notebook

A Magic Notebook jegyzetelésre szolgál, csak éppen webes felületet kell használni toll vagy ceruza helyett. A program bárhol használható, ahonnan a



webkiszolgáló elérhető, és normál módon vagy titkosítással futtatható. A jegyzeteket a saját fájlrendszered HTML-fájlok formájában tárolja, így a webes felület nélkül is megtekintheted őket. Futtatásához webböngésző és olyan webkiszolgáló szükséges, ami kezelni tudja a CGI-parancsfájlokat.

➲ http://www.jonathanscorner.com/etc/magic_notebook

Server Status

Ismét egy program, amivel szemmel tarthatod a kiszolgálókat és állapotukat. Igaz ugyan, hogy X Window kell hozzá, ám használata gyors és egyszerű. Az ilyen programoknak általában szükségük van SNMP-re, a Server Status azonban csak alapvető hálózati szolgáltatásokat igényel. Bármilyen rendszeren futtathatod, amin Perl vagy Tk van, és azonnal jelzi, ha valamelyik fontos szolgáltatásnál gond adódik. Nem értesít levélben, de (a beállítástól függően) 60 másodpercenként önműködően frissül, és könnyen áttekinthető. Futtatásához Perl és a következő Perl-modulok szükségesek: IO::Socket, Tk, Tk::Checkbutton, Tk::Menubutton és Tk::Optionmenu.

➲ http://www.the-den.org/status

Pebrot

Napi munkám során jó néhány kiszolgálón dolgozom, és a többségükre nem telepítek X Windowt, ezért mindig kíváncsi vagyok minden, X-programok helyettesítésére alkalmas parancssori programra. A Pebrot az MSN Messenger Python nyelvű változata, ami a Unix talk programjához hasonlóan nem igényel X Windowt. Ezáltal megkönnyíti a feladatod elvégzését, ha távolról kell futtatnod a programot vagy nincs X telepítve. Futtatásához Python szükséges.

➲ http://pebrot.sourceforge.net

File access statistics

Ez a segédprogram a fájlrendszer tetszőleges részét (vagy egészét) áttekinti, és meglehetősen részletes kimutatást készít a rendszeren található fájlokról. Amennyiben Debiant vagy Debian alapú rendszert (például Knoppixot) futtatsz, további adatokhoz juthatsz a dpkg fájlokkal kapcsolatban. Ez a program a létrehozás vagy módosítás dátuma helyett inkább a hozzáférés ideje alapján határozza meg, hogy az egyes fájlok milyen régiek vagy mennyire elavultak. A már öt éve érintetlenül álló fájlok jó eséllyel történelmi ereklyék vagy kacatok. Futtatásához Perl szükséges.

➲ http://www.hszk.bme.hu/~nm127/file_statistics

Kapcsolódó anyagok a 48-as CD Magazin/Hasznos könyvtárában találhatóak.

Linux Journal 2003. május, 109. szám

David A. Bandel
(dbandel@pananix.com)
Jelenleg Panamában él, Linux- és Unix-tanácsadással foglalkozik. Társszerzője a Que Special Edition: Using Caldera OpenLinux című könyvnek.