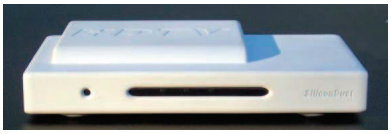
Hírek

HDTV vevő



A ***Silicondust*** 169 dolláros áron piacra dobta ***HDHomeRun*** néven apró ***HDTV*** vevőjét, mellyel akár egyszerre több számítógépen is élvezhető a nagy felbontású ***TV*** adás. Egyszerre két adást képes behangolni, és azt 100 megabites Ethernet porton továbbítja a helyi hálózatra, amelyet például ***VLC***-vel is élvezhetünk. Az eszköz beállításához a mellékelt ***Linuxos (LGPL)***, ***Mac OSX*** vagy ***Windows*** klienst használhatunk.

➲ http://servers.linux.com/servers/07/04/18/1531247.shtml?tid=117&tid=39

Berlin: nem kell nyílt forráskód

Noha ***München*** már megkezdte az átállást nyílt forráskódra, ***Berlin*** nem kér belőle. A német ***Zöld párt*** informatikusa nem érti a döntést, hiszen a nyílt forráskód esetén nem függnének egy amerikai vállalattól, illetve éves szinten a 60 ezres géppark 250 millió eurós informatikai kiadását a felére lehetne csökkenteni. ***München*** előző évben kezdte meg az átállást ***Linux***ra és ***OpenOffice***-ra mintegy 14 ezer munkaállomáson a ***LiMux*** projekt keretében.

➲ http://www.linuxworld.com/news/2007/050307-berlin-says-nein-to-open-source.html

Kis lépés egy embernek...

Három évvel ezelőtt kezdett el ***Michel Xhaard Linux***os meghajtókat írni olcsó webkamerákhoz. Az ok nemes egyszerűséggel: megunta, hogy csak ***Windows*** alatt tudja használni webkameráját. A majd hatvan éves fizikus eddigi meghajtóprogramjaival mintegy 235 webkamera vált használhatóvá ***Linux*** alatt.

➲ http://mxhaard.free.fr/

➲ http://www.theinquirer.net/default.aspx?article=39291

Asztali PC alternatívája

Idén nyáron 250 dolláros áron dobja piacra a kaliforniai ***Zonbu*** a ***Zonbox***-ot. A ***Via C7***-es 1.2 Ghz-es processzorral szerelt csendes számítógép ***Gentoo Linux***szal és két tucat nyílt forrású alkalmazással érkezik. Az eszköz mindössze 15 wattot fogyaszt, szemben a hagyományos ***PC***-k minimálisan 200 wattjával. Az eszköz teljes értékű munkaállomás lehet, hiszen a fél gigabájt memória és a 4 gigabájt ***flash*** tároló mellett hat darab ***USB*** porttal, 10/100-as és ***WIFI*** hálózattal, valamint ***Compact Flash*** kártyahellyel is rendelkezik. Az teljes értékű mivoltát jól példázza az előre telepített alkalmazások listája is.

➲ http://www.linuxdevices.com/news/NS9073106297.html

Újabb mini linux, ami kicsit más...

Az új-zélandi ***Zeljko Aksentijevic*** összeállított egy ***X nélküli***, de ***OpenGL*** fejlesztői környezettel bíró mini ***Linux***ot ***MyOS*** néven. Az ***ISO*** fájl alig 13 megabájt, azonban az ***OpenGL*** fejlesztői környezet „lehántásával" ráfér akár egy floppy lemezre is.

➲ http://one.xthost.info/zelko/opengl.html

➲ http://www.linuxdevices.com/news/NS8925383538.html

Java és Linux a mobiltelefonokon

A ***Sun GPL*** licenccel rendelkező ***Linux*** és ***Java*** alapú rendszert kíván a mobiltelefongyártók kezébe adni. Szemben a jelenlegi megvalósítással, a jövőben szinte minden feladatot ***Java*** látna el, a ***Linux*** csupán a kernelt, az alacsony szintű szolgáltatásokat és függvénykönyvtárakat adja majd.

➲ http://www.linuxdevices.com/news/NS7539760574.html

Pendrive a nyakba

Nem kell többé aggódnunk, hogy elhagyjuk pendrive-unkat. Piacra dobta ugyanis a ***Brando*** 38 dolláros áron legújabb termékét, mely a népszerű nyakpánt (például mobiltelefonhoz vagy belépőkártyához) és egy 2 gigabájtos pendrive keresztezése. A pánt egyedi szöveggel vagy logóval is rendelhető.

➲ http://www.i4u.com/article8741.html

RFID és a magánszféra

Számos helyen alkalmaznak már ***RFID*** azonosítást – útlevél, belépőjegy, tömegközlekedési bérlet, áruvédelmi címke, stb –, melyet azonban elég közelről (maximum pár méter) jogosulatlan személy is leolvashat megfelelő berendezéssel anélkül, hogy az ***RFID*** azonosítót birtokló tudna róla. ***Melanie Rieback***, az amszterdami ***Vrije egyetem*** végzős diákja ***RFID Guardian*** néven kifejlesztett egy eszközt, mellyel elrejthetőek a kíváncsi szemek elől az ***RFID*** kártyák, vagy akár megtévesztő információt is szolgáltathat.

➲ http://www.rfidguardian.org/index.html

➲ http://arstechnica.com/articles/culture/rfid-guardian.ars/1

Silverlight hamarosan Linux alatt is?

Miguel de Icaza, a ***Mono*** projekt vezetője, kijelentette, hogy év végéig elkészítik a ***Silverlight Linux***os verzióját, minthogy az ***.Net*** alapú és így ***Mono***-ra átültetni remélhetőleg nem okoz majd nagy nehézséget. A ***Silverlight*** egy ***Flash***-hez hasonló fejlesztői környezet, mely lehetővé teszi gazdag multimédiás tartalmakkal bíró interaktív oldalak és programok létrehozását.

➲ http://news.com.com/8301-10784_3-9714669-7.html

A Firefox az Opera nyomdokaiba lép?

Mitchell Baker, a ***Mozilla*** alapítvány vezérigazgatója nem tartja kizártnak, hogy hosszú távon a ***Firefox*** megjelenjen a mobiltelefonokon is. Kiemelte, hogy az emberek általában a ***Firefox***ot a jól használható bővítmények miatt kedvelik. Kérdés, hogy ezt hogyan lehetne ezt az előnyt átvinni mobiltelefonokra és hogyan lehetne megjeleníteni az asztali monitorok felbontásához tervezett tartalmat a telefon alacsony felbontású kijelzőjén.

➲ http://apcmag.com/6041/firefox_will_move_to_mobile_phones_mozilla_ceo

Mi használ ennyi memóriát?

Bizonyára gyakran szembesült már a programozó ilyen vagy ehhez hasonló kérdéssel programozás közben. ***Matt Mackall*** éppen ezért egy olyan kernelfolt-gyűjteményen dolgozik, mely valósidejű részletes információt ad a memóriahasználatról. Remélhetőleg ezzel a virtuális memóriakezelő rendszer kevésbé fog fekete doboznak tűnni.

➲ http://www.linuxdevices.com/news/NS4314018608.html

India: 2 megabit ingyen 2009-től

2009-re az indiai kormány szeretné elérni, hogy az indiai állampolgárok ingyenesen érhessék el 2 megabit sávszélességű internet előfizetéseket. Ezzel egyetemben a hagyományos telefonhívásokat is szeretnék átterhelni ***VOIP***-ra.

➲ http://economictimes.indiatimes.com/Broadband_to_go_free_in_2_yrs/articleshow/1955351.cms

Az Adobe forráskódot nyit meg

Év végéig megnyitja az ***Adobe Flex*** környezetének forráskódját, remélve, hogy ez a lépés fejlesztőket csábít el a konkurens ***Microsoft Expression*** és ***Silverlight*** technológiáktól. A ***Flex*** fejlesztői környezet megkönnyíti multimédia tartalmakban gazdag weboldalak létrehozását.

➲ http://www.infoworld.com/article/07/04/26/HNadobeopensourceflex_1.html

Egy terabájt mindenkinek elég lesz?

450 dolláros áron már elérhető a ***Hitachi*** 1 terabájtos 3.5 hüvelykes merevlemeze A ***7K1000***-ben 5 lemez található, oldalanként 100 gigabájt adatot tárol. A 7200 fordulat per perces merevlemez a kornak megfelelő ***SATA2***-es csatolóval érkezik, míg a cache mérete a megszokott 8-16 megabájt helyett 32 megabájt. Az arányokat érzékeltetendő: 8 megabites internetkapcsolat esetén teljes sávszélesség mellett 11 nap folyamatos letöltés esetén telne csak meg.

➲ http://www.linuxdevices.com/news/NS5121840662.html

Flash alapú laptopok

A ***Sony (Vaio Type-G)*** mellett a ***Dell*** is bejelentette, hogy igény esetén hagyományos merevlemez helyett ***flash*** alapú háttértárral is kérheti a vásárló leendő gépét. Ez a ***Latitude D420*** és ***D620*** modelleket érinti, melyekbe 1.8 hüvelykes 32 gigabájtos ***SSD*** meghajtók kerülnek.

A laptop ára 549 dollár, de egyelőre csak Amerikában érhető el.

➲ http://www.infoworld.com/article/07/04/25/HNdellssd_1.html

Nyílt forrású Java

A *SUN* – ígéretének megfelelően – május elején kiadta az addig általa fejlesztett és népszerű *Java* programozási nyelv forráskódját *GPL* licenc alatt, legyen szó akár az *SE*, *ME* vagy *EE* verziókról.

➲ http://www.sun.com/software/opensource/java/

A pingvin szemmel tart

Egycsatornás, teljes felbontású ipari videódigitalizálót mutatott be *Moxa*, mely, 30 képkockát tud továbbítani *Ethernet* hálózaton. Az eszköz *Linux*ot futtat és lehetővé tesz kétirányú hangátvitelt. Tartalmaz továbbá két digitális bemenetet, illetve két relét vezérlési célokra. Az eszköz teljesíti az *IP30*-as szabványt.

➲ http://www.linuxdevices.com/news/NS8880429552.html

Szabad betűkészletek

Szabad betűkészletek érhetőek el a *Redhat*-tól a *Windows*os *Times New Roman*, az *Arial* és a *Courier New* alternatíváiként. A betűkészletek pár megkötéssel, de gyakorlatilag *GPL* jogállással használhatóak.

➲ http://www.press.redhat.com/2007/05/09/liberation-fonts/

Mobil Ubuntu

Matt Zimmerman – a *Canonical* egyik vezetője – bejelentette, hogy hamarosan az *Ubuntu* mobil és beágyazott eszközökre is elérhető lesz. Mobil eszközök között mobiltelefonokat nem, csupán tábla *PC*-k és hasonló eszközöket kell értenünk.

➲ http://www.linuxdevices.com/news/NS2403415870.html

Nano ITX PC

Passzív hűtésű *nano ITX* méretű számítógépet mutatott be a *Damn Small Linux (DSL)* projekt. A rendszer lelke egy *800 Mhz-es Via Eden* processzor

256 megabájt memóriával, amely nem bővíthető. A doboz tartalmaz továbbá két darab *USB 2*-es portot és egy 10/100-as hálózati csatolót is. Ára az igényelt *flash* tárhely méretétől függ. 1 gigabájtos *IDE flash* meghajtóval 475 dollár, fél gigabájtos meghajtóval 399 dollár, flash tárhely nélkül 329 dollár. Persze ettől függetlenül egy 256 megabájtos pendrive-on megkapjuk az előretelepített *Damn Small Linux*ot.

➲ http://www.linuxdevices.com/news/NS4730464966.html

GCC 4.2

Megjelent a *gcc 4.2* változata, melynek talán legfontosabb újdonsága az *OpenMP* – multiprocesszoros programozás – támogatása *C*, *C++* és *Fortran* nyelveken.

➲ http://osnews.com/story.php/17922/GCC-4.2-Released/

Nanotechológia a chipgyártásban

Az *IBM* már a következő generációs chipeken dolgozik, melyek bizonyos mértékben önmagukat építenék fel. Pont úgy, ahogy a természetben az élőlények. Az új technológia használatával a mai litografikus eljáráshoz képest ötöd akkora hely szükséges és az energiaigény is 15 százalékkal csökkenne. A technológia az *IBM* reményei szerint akár 3-5 éven belül elterjedhet, de az első chipre 2009-ig várni kell.

➲ http://www.pcmag.com/article2/0,1895,2125715,00.asp

➲ http://news.bbc.co.uk/1/low/sci/tech/3301025.stm

Linux a zsebben

A párizsi *Linutop*-tól már elérhető a *Linutop*, mely egy apró, mindösszesen 6 watt fogyasztással bíró, de mégis szinte teljesértékű *PC*. A méretei – 9.3 x 2.7 x 15 centiméter – a 433 Mhz-es *AMD Geode* processzor mellett tartalmaz 256 megabájt memóriát. Háttértárként a hozzákapott 1 gigabájtos pendriveot használhatjuk.

Teljesértékű, hiszen rendelkezik *VGA* kimenettel, 10/100-as hálózati csatolóval, valamint 4 darab *USB* 2-es aljzat is helyet kapott rajta, melyen keresztül akár *wifi*vel is bővíthető. A rendszer *Xubuntu*t futtat. Darabja 280 euró, de a nyolcas csomagot már 2100 euróért megkapjuk.

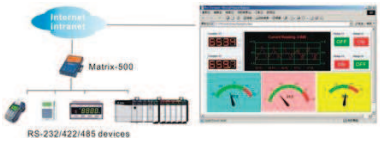
➲ http://www.linuxdevices.com/news/NS3892860033.html

Áttörés várható az akkumulátorok technológiájában

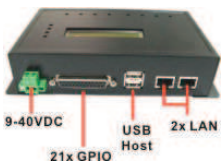
Az *Argonne National Labs* egy új – egyelőre kísérleti – akkumulátor fajtát mutatott be, melynél a lítium ion akkumulátorban a hagyományos elektródákat mangánnal helyettesítették. Ennek köszönhetően az akkumulátor mintegy *kétszer annyi energiát képes tárolni* (250 mAh/g). A technológia még nem áll készen a bevetésre, hiszen számos problémát kell megoldani: például túl gyorsan használódik el – körülbelül 60 töltés –, illetve kisütés közben oxigén termelődik, amit el kell vezetni. Az viszont mellette szól, hogy a jelenlegi technológiánál olcsóbb lenne mangánt alkalmazni.

➲ http://arstechnica.com/news.ars/post/20070508-manganese-electrode-could-double-litium-ion-battery-capacity.html

Ipari automatizálás magasfokon



Az ***Artila Electronics*** elkészítette ***ARM9*** alapú ***Matrix-520***-as egységét, melyet elsősorban ipari automatizálásra ajánlanak (hűtés, fűtés, liftek vezérlése, stb.) 144x32 képpontos pontmátrix kijelzőjén helyben is megjeleníthet adatokat, azonban 2 darab hálózati csatolójával akár a világ túlvégéről is elérhető. Ezenkívül 2 darab ***USB*** porttal, 8 darab nagy sebességű soros porttal (ebből 5 db tudja az ***RS232***-n kívül az ***RS422/485***-öt is), illetve 21 darab programozható digitális be/kimenettel is bír.



➲ http://www.linuxdevices.com/news/NS8159964573.html

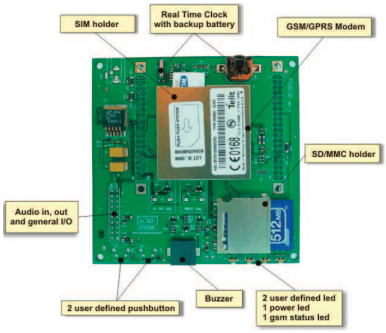
Linux az utcai lámpákon

A ***Harvard*** egyetem kutatói a ***BBN Technologies*** munkatársaival karöltve a massachusettsi ***Cambridge***-ben az utcai lámpákra 100 darab, beágyazott ***Linuxot*** futtató számítógépet szerel fel, melyek egymással ***wifi*** kapcsolaton fognak kommunikálni. A hálózat célja kezdetben időjárási és légszennyezettségi adatok gyűjtése lesz.

➲ http://www.cio.com/article/108413/Harvard_BBN_Use_Streetlamps_to_Light_Up_Wireless_Network

SMS-TCP/IP átjáró

Linux alapú ***SMS-TCP/IP*** átjárót mutatott be az ***Acme Systems***. A ***FoxBox*** négysávos ***GSM*** modemet, web és email szolgáltatásokat kínál, valamint helyben tudja tárolni az üzeneteket. A rendszer lelke egy ***Axis Eltrax 100LX RISC*** processzor – mely 100 millió utasítást hajt végre másodpercenként –, valamint 4 megabájt ***flash*** memória és 32 megabájt ***RAM***.



A 10/100-as hálózati csatoló mellett a két ***USB 1.1***-es porton keresztül ***Bluetooth*** vagy ***wifi*** adapterekkel is bővíthetjük. Az eszköz közvetlenül az ***Acme Systems***-től rendelhető 750 euróért.

➲ http://www.linuxdevices.com/news/NS2761247659.html

Akár egy terapixel

Az ***Aperio Technologies*** már korábban kifejlesztett ***bigTIFF*** formátumával lehetséges volt nagy felbontású képeket tárolni. Ennek forráskódját nyílttá tették, hogy még gyorsabban elterjedhessen. A honlapon található demonstrációs céllal egy terapixeles kép, mely 143 gigabájtot foglal a merevlemezen.

➲ http://www.aperio.com/bigtiff/

Nano méretű fényforrás

A ***Craighead kutatócsoport*** a ***Corenll*** egyetemen bemutatta az eddigi legkisebb – ember alkotta – fényforrást. A nanoszál alig ***250 nanométer*** (hajszál vastagságnak töredéke) átmérőjű és 600 nanométer hullámhosszú fényt (naracssárga szín) bocsát ki 3-4 Voltos feszültség hatására, azonban ezt 100 Voltra emelve olyan fényerőt adott, hogy szabad szemmel is láthatóvá vált sötét szobában. A szálak ***polimer***ből készültek, így akár nagyon vékony hajlékony kijelzőként is találkozhatunk velük a jövőben.

➲ http://www.justchromatography.com/general/nano-light-bulbs

Medve Zoltán
(e-medve@e-medve.hu)
2001-ben kezdett „Linuxolni”, de már korábban is ismerkedett a szabad szoftverek világával. Ha éppen nem a gép előtt ül, akkor fotózgat, olvasgat vagy bicajozik.

Mi újság a rendszermag fejlesztése körül

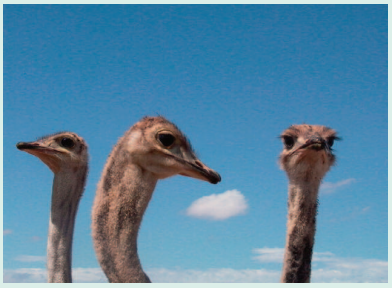
Molnár Ingó létrehozott egy olyan új, ***Sylets***-nek nevezett, nyelvszerű burkolót, amelynek segítségével felhasználói térből lehet rendszerhívásokat kiadni. Az ezen a nyelven írt miniprogramok a kerneltérből való kilépés nélkül képesek aszinkron módon futtatni a rendszerhívásokat és a felhasználó igényeinek megfelelően lereagálni azok eredményét. A ***Syslets*** burkolók használatával ***Ingo*** 33,9 %-os sebességnövekedést mért gyorstárazott (***cached***) szinkron ***I/O*** műveleteknél, míg az ugyanilyen de nem gyorstárazott műveletek esetében 19,2 százalékos volt a teljesítmény növekedése. A kernelfejlesztők körében a ***Syslets*** nyelv igen komoly érdeklődést váltott ki, bár ***Linus Torvalds*** szerint a programozási felület túlságosan összetett, ami nagyban gátolja, hogy az eseti felhasználók kísérletezni tudjanak vele. Összességében tehát a ***Syslets*** rendszernek még fejlődnie kell ahhoz, hogy bekerülhessen a fő forrásfába.
Az ***Intel*** elkészítette a ***PRO/Wireless 3945ABG*** típusú hálózati csatolók meghajtóját. Az egyéb ***Intel*** által készített meghajtóktól eltérően ez a kód nem támaszkodik semmiféle, zárt forrású démonra, vagyis teljesen nyílt forrású. Ugyanakkor a futtatásához szükség van a mikrokód frissítésére. A licenceléssel kapcsolatos ilyetén fejlődés amúgy láthatólag nem valamiféle a hardver terén történt fejlesztésnek, hanem a hatékonyabb mikrokódnak köszönhető. Az ***Intel*** új meghajtóját a fejlesztői közösség jól fogadta, vagyis a legjobb úton halad a forrásfába való beillesztés felé.
A sokak által jól ismert, és nagy múltra visszatekintő ***\0 loophole*** szolgáltatás, amely lehetővé tette, hogy szükség esetén a valójában nem a ***GPL*** hatálya alá tartozó meghajtók is ***GPL*** jogállásúnak tüntessék fel magukat, hamarosan kikerül a rendszermagból. Néhány fejlesztő, akik közül eddig ***Jan Engelhardt*** mutatkozott a legaktívabbnak már be is nyújtotta a lyukat befoltozó javítást. Ugyanakkor fontos megemlíteni, hogy ezzel a folttal kapcsolatban meglehetősen sok az ellentmondás. Egyesek szerint ha a kernel komolyabb korlátozásokat érvényesít a nem ***GPL***-es mint a ***GPL***-es meghajtókkal kapcsolatban, azzal egyben maga is megsérti a ***GNU General Public License***-ben leírtakat, hiszen kikényszeríti egy adott licenctípus érvényesülését. Amíg megvan a kódban az említett kiskapu, addig ez a vita csendben meghúzódik a mélyben, hiszen a meghajtók fejlesztői kikerülhetik a nem ***GPL*** kódokra vonatkozó megszorításokat. Ha azonban kikerül a rendszermagból, akkor az olyan cégek – ilyen például a ***LinuxAnt*** – amelyek a múltban vastagon kihasználták a kerülőutat azonnal tiltakozni kezdenek majd.
A ***KVM*** virtuális gép kódját a fejlesztők átemelték a ***Subversion*** rendszer alól egy ***git*** tárba. Ennek a döntésnek több oka is volt. ***Avi Kivity*** az indokok között megemlítette, hogy a ***Subversion*** képtelen volt a teljes kernelfát hatékonyan kezelni és tárolni, valamint hogy a fejlesztők a kód egyes, általuk felügyelt ágait függetlenül szerették volna fejleszteni és karbantartani.
Jon Masters átvette a ***module-init-tools*** fejlesztésének koordinálását ***Rusty Russeltől***, és ezt immár a ***MAINTAINERS*** fájlban is feltüntette. ***Evgeniy Dushistov*** szintén létrehozott ebben a fájlban egy ***USF*** bejegyzést, és abban feltüntette magát fejlesztőként.
Alessandro Di Marco gyakorlatilag számára is váratlan sikert aratott a közelmúltban, amikor közzétett egy új felhasználói inaktivitást jelző ***triggert***, amivel korábban csak úgy szórakozásból foglalkozott. Ez az ügyes kis szolgáltatás azt tudja, hogy ha a felhasználó egy bizonyos ideig nem csinál semmit, akkor kiad egy megfelelő ***ACPI*** jelet. ***Alessandro*** igazából a móka kedvéért kezdett ezzel a témával foglalkozni és teljesen szándékosan kerülte eddig a megvalósítással kapcsolatos kérdések feltevését, mivel úgy gondolta, hogy rajta kívül nem sok élő ember lehet, akit ez a téma izgatna. Alaposan tévedett, ugyanis mint most kiderült, rengeteg fejlesztőt fogott meg az ötlet, és számos a megvalósítással kapcsolatban hasznosnak tűnő javaslat is született.
Az elsők között mutatott rá ***Arjan van de Ven***, hogy az ***uevent*** mechanizmus sokkal hatékonyabb lenne a probléma megoldására, mint az ***ACPI***. ***Pavel Machek*** szerint az ***Alessandro*** új kódja által a ***/proc*** fájlrendszerben létrehozott fájl sokkal jobb helyen lenne a /sys könyvtárban. Szintén ***Pavel*** jegyezte meg, hogy az ilyen szolgáltatásokat inkább a felhasználói térben kellene megvalósítani, ám ***Alessandro*** szerint ez elég alaposan megbonyolítaná a kódot. A szerző gyorsan reagált a legtöbb egyéb javaslatra is, megírt egy új változatot, és igyekezett megválaszolni minden, a kernelfejlesztők levelezési listáján felmerült kérdést.
Megkezdődtek a következő kernelfejlesztői csúcs (***Linux Kernel Summit***) előkészületei. A legújabb hír, hogy a társaság a kanadai ***Ottawa*** helyett most az angliai ***Cambridge***-ben fog összegyűlni. Az új helyszín választása amúgy felvetette más jövőbeni helyszínek lehetőségét is, melyek között egyelőre ***Ausztria***, ***India*** és a ***Cseh Köztársaság*** szerepel. A hely megválasztásánál a fő szempont a rendezvény teljes költsége. Igaz ugyan, hogy a kernelfejlesztők közül sokan dolgoznak olyan cégeknél, amelyek minden évben fizetik a repülőjegyüket a konferenciára, de ettől függetlenül előfordulhatnak bizonyos helyeken olyan árak, amelyek egyesek számára egyszerűen megfizethetetlenek. Szintén szempont, hogy ha egy országból eleve sok fejlesztő kerül ki, akkor abban az országban nagyobb valószínűséggel fogják megrendezni az összejövetelt, legalábbis ezt állítja ***Theodor Y. Ts'o***, az egyik főszervező. Ettől persze még bármi egyéb is történhet.

Zack Brown (LJ. 158)

Hurrá, nyaralunk!

Elérkezett az idő, mikoron annyira elhatalmasodik a tunyaság és a kényelmesség az ember fián és lányán, hogy már a munkahelyén sem tudja kipihenni magát. Ekkor elhatározza, hogy tömegesen országot cserél; idegen földön szórja el megspórolt pénzét, s mindezt mérhetetlen mennyiségű fényképpel dokumentálja.

Bizony, ha már kezdünk úgy kinézni, mint az útlevélképünk, akkor igazán szükségünk van a nyaralásra. Az utazáshoz jótanácsokat is kaphatunk, például: fejlődő országokban ne igyunk vizet, fejlett országokban pedig ne vegyünk levegőt. Ugyan meglehet, bőröndünkbe kell csomagolnunk az oxigénpalackot és a teli kulacsot, ellenben némely



egzotikus kiránduláson saját szemeinkkel ellenőrizhetjük az élelmiszerek eredetét és minőségét.
Történhetne ez egy zuluföldi bölényszafarin is illegális vadászaton, de aki beéri „szerényebb" vacsorával, az végignézheti egy struccbébi tojásból kikecmergését, majd a (sült) mama vagy papa tányéron felszolgálását. Erősebb idegzetűek a strucc-barbecue emésztését serkenthetik némi gepárd-, illetve oroszlánkergetőzéssel valamely afrikai parkban, mások ez idő alatt elcserélhetik nejüket egynehány tehénre, kecskére vagy turmixgépre. Aki vízre vágyik, gondolázhat krokodilok és vízilovak között, esetleg merülhet cápákkal, avagy fürdőzhet pingvinek társaságában. Ehhez nem kell útitársul bérelnünk egy madarat az állatkertből (ha mégis, ne feledjük, már a kisállatoknak is dukál az útlevél), és nem is kell feltétlenül csatlakoznunk egy déli-sarki expedícióhoz. Manapság az utazási irodák egy kis szörfözést vagy raftingolást sem kínálnak fehércápák vagy pingvinek nélkül.
Akinek nem telik afrikai vagy újzélandi vadregényes kiruccanásra, még mindig pótolhatja az izgalmakat egy hazai bungee-jumping utáni Zoo-látogatással. De hogy senki se maradjon ki a nyári pingvin-kalandból: gyermeteg lelkületűeknek kínálják a Pancsi Pingvint, mely egyszemélyes társasjáték felhúzás után a fürdőkádban énekel, sőt merülni is tud, s közben bugyborékolva danolászik tovább. Kéretik az erről szóló fényképes dokumentációt nem beküldeni a szerkesztőségbe!

Halusz Léna

Bevetés közben – Ismerkedés az Ajaxszal

Hogy kerül az A – mint aszinkron – az Ajaxba?

Sok programozó, így én is, jó ideje ismeri a *Javascript*-et, mellyel dinamikusan módosíthat *HTML* oldalakat. Persze más apró feladatra is alkalmas, ilyen például az űrlap érvényességének ellenőrzése. Az utóbbi években azonban a *Javascript* lett a húzóerő az alkalmazásfejlesztők körében, köszönhetően az *Ajax*-os megoldásoknak.
A *Javascript* népszerűsége előtt egy-az-egyben megfeleltetés volt a felhasználó reakciója és a *HTML* oldal megjelenítése között. Ha ráklikkelt a felhasználó egy linkre, akkor az aktuális oldal eltűnt és betöltődött az új. Ha elküldött egy űrlapot, akkor azt megkapta a webkiszolgáló és a választ jelenítette meg. A hagyományos webalkalmazásoknál szerveroldalon dolgozták fel a felhasználói adatokat és a szerver oldalon rakták össze a dinamikus oldalakat is.
Az *Ajax*-os alkalmazások megosztják a terhelést azzal, hogy nagyobb szerepet kap a kliens oldali *Javascript*. Számos *Ajax* program teljes *HTML* oldalt generál le, amely aztán egy-az-egyben jelenik meg a webböngészőben. A többinél azonban a szerver csupán apró *XML* formátumú morzsákat ad a kliensnek. Ezt a kliens kéri és dolgozza fel *Javascript*-tel, majd pedig frissíti a *HTML* oldal egy részét anélkül, hogy a teljes oldalt újra kellene tölteni vagy le kellene cserélni. A webes *DOM* (*document object model*) és *CSS* (*cascading stylesheets*) segítségével az *Ajax*-os alkalmazások ugyanazokkal a tulajdonságokkal bírnak – használhatóság, felhasználóbarát, azonnali válasz –, mint azt az asztali alkalmazásoknál megszokta a felhasználó.
Az elmúlt pár hónap után most folytatjuk a kliens oldali *Javascript* és *Ajax* felfedezését. Az előző hónap témája a felhasználók webes regisztrációja volt. Noha a tényleges regisztráció szerver oldali, olyan megoldás kerestünk, ami *Ajax* segítségével figyelmezteti a felhasználót, ha az adott azonosító már foglalt. Természetesen szerver oldali ellenőrzést is használhatnánk, ez azonban az oldal frissítésével jár, ami időbe telik.
A múlt hónapban bemutatott megoldás a felhasználó számára megfelelő (különösen, ha kedveli a spártaian egyszerű kinézetet). Azonban a problémát nem igazán *Ajax*-os módon oldottuk meg. Egy tömbben beégetve tároltuk a felhasználói neveket és ebben a tömbben kerestünk. A megvalósítás számos sebből vérzik, hiszen bárki megtekintheti a már regisztrált felhasználók azonosítóit, illetve sok felhasználó esetén a tömb óriásira dagad, így sokáig tart letölteni az oldalt. Az oldal letöltési ideje és az abban való keresés időtartama a regisztrált felhasználók számának növekedésével arányosan nő.
Ezek a problémák *Ajax*-os megoldással elkerülhetők. A felhasználói lista *Javascript* forrásba drótozása és a teljes lista lekérése helyett csupán megkérdezzük a szervertől, hogy az adott felhasználó létezik-e már? Ez elég gyors letöltést és reakcióidőt eredményez amellett, hogy áttekinthetőséget és egyszerű bővíthetőséget ad.
Most belevetjük magunkat az *Ajax*-ba. A korábbi szerver és kliens oldali programot módosítjuk úgy, hogy aszinkron módon kérje le a felhasználóneveket. Eközben látni fogjuk, milyen egyszerű *Ajax* alkalmazást készíteni, vagy egy meglévő web alkalmazást felruházni *Ajax*-os funkciókkal. A cikk végére a kedves Olvasó is képes lesz hasonló szerver- és kliens oldali *Ajax* alkalmazásokat írni.

Ajax hívás

A *Javascript*-es *XMLHttpRequest* objektum teszi lehetővé az *Ajax*-os lehetőségek nagy részét. Ennek az objektumnak a segítségével indíthat *HTTP* kérést egy *Javascript* függvény és ennek segítségével reagálhat rá. (Biztonsági okok miatt az *XMLHttpRequest* objektum csak azzal a szerverrel tud adatot cserélni, amelyről letöltődött az adott oldal.) A *HTTP* kérés egyaránt lehet *GET* vagy *POST*, azonban az utóbbival tetszőleges hosszúságú és bonyolultságú kéréseket indíthatunk.

A legérdekesebb és egyben az ***Ajax*** paradigma alapja, hogy az ***XMLHttpRequest*** indíthat ***szinkron*** (a böngészőnek várnia kell, amíg a teljes válasz megérkezik) és ***aszinkron*** (a böngésző használható, miközben érkeznek az adatok) ***HTTP*** kéréseket is. Az ***Ajax*** rendszerint aszinkron hívásokkal dolgozik. Ez lehetővé teszi, hogy a weboldal egyes részeit egymástól függetlenül frissítsük, voltaképpen egy időben válaszol többszörös adatbevitelre.
Elméletileg az alábbi ***Javascript*** sorral hozhatunk létre egy példányt az ***XMLHttpRequest*** objektumból:

```
var xhr = new XMLHttpRequest();
```

Sajnos az élet nem ilyen egyszerű. Azért nem, mert a legtöbb ember ***Internet Explorer***t használ. Az ***Explorer*** nem rendelkezik beépített ***XMLHttpRequest*** objektummal, így nem is példányosítható az említett módon. Így azonban igen:

```
var xhr = new ActiveXObject
➥("Msxml2.XMLHTTP");
```

Álljunk csak meg! Néhány ***Explorer*** verziónál ez azonban kicsit másképp néz ki:

```
var xhr = new ActiveXObject
➥("Microsoft.XMLHTTP");
```

Hogyan kezeljük le ezt a három különböző ***XMLHttpObject*** példányosítást? Az egyik megoldás: detektáljuk szerver oldalon a böngészőt. Persze ugyanez kliensoldalon is megoldható. De a legelegánsabb módra, amivel mostanáig találkoztam – ***Michael Mahemoff: Ajax Design Patterns*** című könyvében leltem. ***Mahemoff*** a ***Javascript*** kivételkezelését használja, amíg valamelyik nem lesz jó. A három megoldást egy függvénybe helyezve és ezt rendelve az `xhr` változóhoz, biztosíthatjuk alkalmazásunk keresztplatfromos működését:

```
function getXMLHttpRequest () {
try { return new ActiveXObject
➥("Msxml2.XMLHTTP"); }
➥catch(e) {};
try { return new ActiveXObject
➥("Microsoft.XMLHTTP"); }
➥catch(e) {}
try { return new XML
➥HttpRequest(); } catch(e) {};
    return null;
}
var xhr = getXMLHttpRequest();
```

A fenti részlet futtatásakor az `xhr` értéke vagy `null` (ez azt jelenti, hogy nem tudta példányosítani az `XMLHttpRequest` objektumot) vagy pedig az `XMLHttpRequest` objektum az adott böngészőben megfelelő megvalósítása. A példányosítás után már van egy keresztplatformos `XMLHttpRequest` objektumunk, így továbbá nem kell figyelnünk rá.
A leggyakoribb eljárás az `open`. Ez inicializálja a ***HTTP*** kérést egy bizonyos ***URL***-re a forrás szerver felé. Jelen esetben például így hívhatjuk meg az `xhr` példányunk `open` eljárását:

```
xhr.open("GET", "foo.html",
➥true);
```

Az első paraméter (***GET***) megmondja az `xhr.open`-nek, hogy ***GET*** típusú ***HTTP*** lekérdezés lesz. A második paraméter megadja az ***URL***-t, amit szeretnénk elérni. Jegyezzük meg, minthogy a forrás kiszolgálójához csatlakozunk, a protokoll és a szerver címe hiányzik. A harmadik paraméterrel megadjuk, hogy ***aszinkron*** (`true`) vagy ***szinkron*** (`false`) kérést szeretnénk-e indítani? Csaknem minden ***Ajax*** alkalmazásban `true` az értéke, ami azt jelenti, hogy a böngészőben megnyitott oldal addig is használható, amíg a ***HTTP*** kérés válaszát várja.
Az aszinkron ***HTTP*** az ***Ajax*** fő vonzereje. Minthogy a ***HTTP*** kérés nem befolyásolja a felhasználói felületet, így a webes alkalmazás sokkal inkább helyi, asztali alkalmazásnak tűnik.
Az `xhr.open()` még nem jelent ***HTTP*** kérést, csupán beállítja az objektumot, a küldéskor az itt beállított paramétereket fogja használni. A kérés elküldésére ezt használjuk:

```
xhr.send(null);
```

Az `XMLHttpRequest` soha nem ad vissza ***HTTP*** választ, akárki hívja is meg a `xhr.send()` eljárást. Ennek oka, hogy az `XMLHttpRequest`-et aszinkron módban használjuk, hiszen az `xhr.open()`-nél true-ra állítottuk ezt a paramétert. Nem tudjuk megjósolni, hogy fél másodpercen, öt másodpercen, 1 percen vagy 10 órán belül kapunk választ. Ehelyett, megmondjuk a ***JavaScript-nek***, hogy mely függvényt hívja meg, ha megérkezik a válasz. Ez a függvény fogja beolvasni és feldolgozni a választ, illetve ez hajtja végre a szükséges utasításokat.
A `parseHttpResponse` egyszerű megvalósítása így néz ki:

```
function parseHttpResponse() {
    alert("entered parseHttp
➥Response");
    if (xhr.readyState == 4) {
        alert("readystate ==
➥4");
        if (xhr.status == 200) {
            alert
➥(xhr.responseText);
        }
        else
        {
            alert("xhr.status
➥== " + xhr.status);
        }
    }
}
```

A `parseHttpResponse` meghívásra kerül, ha megérkezik a válasz az ***Ajax***-os kérésünkre. Hogy biztosak legyünk, megérkezik-e a teljes válasz, figyeljük a `xhr.readyState` attribútumát. Ha ez ***4***, akkor kapott az `xhr` választ. Következő lépésben ellenőrizzük a válasz ***HTTP*** kódját. A sikeres ***(OK)*** lekérésnek a ***kódja 200***. Természetesen kaphatunk ***404-et ("file missing")*** a szervertől, de az se kizárt, hogy egyáltalán nem tudtunk kommunikálni a szerverrel. Ahhoz, hogy a ***JavaScript*** meghívja a `parseHttpResponse` függvényünket ***HTTP*** válasz esetén, állítsuk át az `XMLHttpRequest` objektum `onreadystatechange` attribútumát:

```
xhr.onreadystatechange =
➥parseHttpResponse;
```

Végül, miután megbizonyosodtunk róla, hogy megkaptuk a választ és minden rendben, a szöveget a `xhr.responseText` eljárással nyerhetjük ki. Az ***XMLHttpRequest***-től kétféle formátumú adatot kaphatunk: egyszerű szöveg (mint itt is) vagy ***XML*** dokumentum. Utóbbi esetben használhatjuk a ***DOM***-ot navigációhoz, akárcsak egy weboldal esetén.

1. Lista ajax-test.html

```
<!DOCTYPE html PUBLIC “-//W3C//DTD XHTML 1.0 Strict//EN”
“http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd”>
<html xmlns=”http://www.w3.org/1999/xhtml”>
  <head><title>Ajax test</title>
    <script type=”text/javascript”>
    function getXMLHttpRequest () {
    try { return new ActiveXObject(“Msxml2.XMLHTTP”); }
    ➥catch(e) {};
    try { return new ActiveXObject(“Microsoft.XMLHTTP”); }
    ➥catch(e) {}
    try { return new XMLHttpRequest(); } catch(e) {};
        return null;
    }
    function parseHttpResponse() {
        alert(“entered parseHttpResponse”);
        if (xhr.readyState == 4) {
            alert(“readystate == 4”);
            if (xhr.status == 200) {
                alert(xhr.responseText);
            }
            else
            {
                alert(“xhr.status == “ + xhr.status);
            }
        }
    }
    var xhr = getXMLHttpRequest();
    alert(“xhr = “ + xhr);
    xhr.open(“GET”, “atf.html”, true);
    xhr.onreadystatechange = parseHttpResponse;
    xhr.send(null);
    </script>
  </head>
  <body>
    <h2>Headline</h2>
    <p>Paragraph</p>
  </body>
</html>
```

Természetesen egy *Ajax* alkalmazás nem hívja meg minden lépésben az `alert` függvényt. Helyette valami hasznosabban csinál: szöveget módosít vagy a dokumentumfához ad ágakat vagy töröl belőle, de akár a kinézetet is módosíthatja. A forráskód az *1. Listában* olvasható.
Noha az ***ajax-test.html*** egyszerű, mégis egy teljes értékű *Ajax* program. A kipróbáláshoz szükségünk van a webszerveren a `DocumentRoot` könyvtárban az ***atf.html*** állományra. (Különben ***404-es HTTP*** hibakódot kapunk.) Ha felmerült a kérdés, vajon mennyire lehet bonyolult egy *Ajax* hívás, akkor a példa mutatja, hogy viszonylag egyszerű.

Ajax-os regisztráció

Most, hogy láttuk, hogy működik egy *Ajax*-os program, a tudás birtokában módosítsuk a múlt havi regisztrációs programunkat. A korábbi megoldásnál a *JavaScript*-ben definiáltuk a felhasználói azonosítókat. Ha a felhasználó egy olyan azonosítót kért, ami már jelen volt a rendszerben, úgy jelezte azt és nem engedte a regisztrációt. Nem írom le az összes problémát ezzel a megközelítéssel kapcsolatban, mert sok volt. Egyszerű alternatívaként mi lenne, ha *Ajax*-al érnénk el a felhasználói azonosítók listáját? Ebben az esetben biztosak lehetünk, hogy naprakész a lista.
Mi lenne, ha a fix, előre bedrótozott lista helyett a webszerverről kérnénk le azt? (Természetesen ez nem olyan kulturált megoldás, mintha igent vagy nemet kapnánk egy bizonyos felhasználói névre. Arról a következő hónapban beszélek.) Ha az *Ajax*-os lista dinamikusan generálódna, akkor a szükséges adatokat adatbázisból is kinyerhetnénk. Ezt *XML*-be elküldve egyszerűen betölthető lenne a tömbbe. Hogy a mostani példánkon egyszerűsítsek, statikus oldalt használunk dinamikus helyett. Természetesen ha a kedves Olvasó korábban már írt szerveroldali webalkalmazásokat, úgy semmiség a statikus fájlt dinamikussá alakítani.
A regisztrációs oldal, amely ezt a listát értelmezi, alább található. Az ***ajax-register.html*** fájl hasonló a múlt havihoz. A nem *Ajax*-os megoldásnál tömbben (`usernames`) tároltuk a felhasználói azonosítókat. Definiáltuk a `checkUsername` függvényt, melyet a `username` szöveges mező `onchange` kezelőjéhez rendeltünk hozzá.

2. Lista usernames.txt

```
abc
def
ghi
jkl
mno
pqr
stu
vwx
yzz
```

A függvény így akkor hívódott meg, ha a felhasználó befejezte a kívánt felhasználói név begépelését. Ha az már regisztrálva volt, úgy a felhasználó kapott egy figyelmeztetést és a véglegesítő gomb inaktívvá vált. Különben a felhasználó elküldhette az adatot a szerveroldali alkalmazásnak, jelezve hogy szeretne regisztrálni.
Hogy a múlt havi regisztrációs oldalt *Ajaxossá* alakítsuk, módosítjuk a `checkUsername` függvényt. Ez akkor hívódik meg, ha a felhasználó befejezte a felhasználói név bevitelét.

3. Lista ajax-register.html

```
<!DOCTYPE html PUBLIC “-//W3C//DTD XHTML 1.0
➥Strict//EN”
“http://www.w3.org/TR/xhtml1/DTD/xhtml1-
➥strict.dtd”>
<html xmlns=”http://www.w3.org/1999/xhtml”>
  <head><title>Register</title>
    <script type=”text/javascript”>
    function getXMLHttpRequest () {
        try { return new ActiveXObject
➥(“Msxml2.XMLHTTP”); } catch(e) {};
        try { return new ActiveXObject
➥(“Microsoft.XMLHTTP”); } catch(e) {};
        try { return new XMLHttpRequest(); }
➥catch(e) {};
        return null;
    }
    function removeText(node) {
        if (node != null)
        {
        if (node.childNodes)
        {
            for (var i=0 ; i <
➥node.childNodes.length ; i++)
            {
            var oldTextNode = node.childNodes[i];
            if (oldTextNode.nodeValue != null)
            {
                node.removeChild(oldTextNode);
            }
            }
        }
        }
    }
        function appendText(node, text) {
            var newTextNode =
➥document.createTextNode(text);
            node.appendChild(newTextNode);
        }
        function setText(node, text) {
            removeText(node);
            appendText(node, text);
        }
        var xhr = getXMLHttpRequest();
        function parseUsernames() {
            // Set up empty array of usernames
            var usernames = [ ];
            // Wait for the HTTP response
        if (xhr.readyState == 4) {
        if (xhr.status == 200) {
            usernames =
➥xhr.responseText.split(“\n”);
        }
        else
        {
            alert(“problem: xhr.status = “ +
➥xhr.status);
        }
        }
            // Get the username that the person
➥wants
            var new_username =
➥document.forms[0].username.value;
            var found = false;
            var warning = document.getElementById
➥(“warning”);
            var submit_button =
➥document.getElementById(“submit-button”);
        // Is this new username already taken?
Iterate over
        // the list of usernames to be sure.
            for (i=0 ; i<usernames.length; i++)
            {
                if (usernames[i] == new_username)
                {
                    found = true;
                }
            }
        // If we find the username, issue
➥a warning and stop
        // the user from submitting the form.
            if (found)
            {
                setText(warning, “Warning:
➥username ‘“ + new_username
                +”’ was taken!”);
                submit_button.disabled = true;
            }
            else
            {
                removeText(warning);
                submit_button.disabled = false;
            }
        }
        function checkUsername() {
            // Send the HTTP request
        xhr.open(“GET”, “usernames.txt”, true);
        xhr.onreadystatechange = parseUsernames;
        xhr.send(null);
        }
    </script>
  </head>
  <body>
    <h2>Register</h2>
    <p id=”warning”></p>
    <form action=”/cgi-bin/register.pl”
➥method=”post”>
        <p>Username: <input type=”text”
➥name=”username”
onchange=”checkUsername()” /></p>
        <p>Password: <input type=”password”
➥name=”password” /></p>
        <p>E-mail address: <input type=”text”
➥name=”email_address” /></p>
        <p><input type=”submit” value=”Register”
➥id=”submit-button”/></p>
    </form>
  </body>
</html>
```

Az előredefiniált tömb helyett a `checkUsername` *Ajax*-os kérést indít a szerver felé. A múlt havi nem *Ajax*-os változathoz képest most csupán ennyit csinál a `checkUsername`. A frissített függvény így néz ki:

```
function checkUsername() {
xhr.open("GET",
➥"usernames.txt", true);
xhr.onreadystatechange = parse
➥Usernames;
xhr.send(null);
}
```

Ahogy látható, lekéri a *usernames.txt*-t a szerverről. Ha az `xhr` állapota megváltozik, meghívásra került a `parseUsernames` függvény. A függvényt komoly dolgokkal vérteztük fel. Először is a kapott állományt tömbbé alakítjuk:

```
var usernames = [ ];
if (xhr.readyState == 4) {
if (xhr.status == 200) {
    usernames =
➥xhr.responseText.split("\n");
}
}
```

Itt ismét belebotlunk a korábbi *Ajax*-os példába: várunk, amíg az `xhr.readyState` értéke *4* lesz, majd leellenőrizzük a `xhr.status`-t (a *HTTP* válaszkódja) hogy *200*-e? Itt már tudjuk, hogy hibátlanul megkaptuk a *usernames.txt* fájl tartalmát. Ez tartalmazza a már regisztrált felhasználók listáját. Egy felhasználói név egy sorban, amint az a 2. Listában is látszik. A *JavaScript* `split` függvényét használva a `usernames` tömbbe helyezzük a felhasználói neveket.
Innentől már használhatjuk a múlt havi nem *Ajax*-os megoldást. Először is a *DOM* segítségével lekérdezzük pár elem azonosítóját:

```
var new_username =
➥document.forms[0].username.
➥value;
var found = false;
var warning = document.
➥getElementById("warning");
var submit_button = document.
➥getElementById
➥("submit-button");
```

Ezután megnézzük, hogy az éppen begépelt felhasználói név szerepel-e a tömbünkben:

```
for (i=0 ; i<usernames.length;
➥i++)
{
    if (usernames[i] ==
➥new_username)
    {
        found = true;
    }
}
```

Ha szerepel, akkor figyelmeztető üzenetet írunk az oldal tetejére. Különben töröljük az esetleges figyelmeztetéseket:

```
if (found)
{
    setText(warning, "Warning:
➥username '" + new_username
➥+"' was taken!");
    submit_button.disabled =
➥true;
}
else
{
    removeText(warning);
    submit_button.disabled =
➥false;
}
}
```

Nézzük csak, megfelelő módon kezeljük a felhasználói neveket? Nem igazán, hiszen most még csak nagyon kezdetleges módon implementáltuk az *Ajax*-ot. A hatékonyságon és a biztonságon javíthatunk.
Az egyik probléma a statikus fájl. Természetesen a szerveren a *cron* segítségével időnként legeneráltathatnánk a *usernames.txt* állományt, ez azonban eléggé fapados megoldás. Helyette használhatunk szerveroldali programot adatbázis lekérdezéssel megtámogatva. Statikus oldalról dinamikusra váltani már csak a teljesítmény javítása miatt is jó ötlet.
Biztonsági okok is vannak. A múlt havi programunkkal a teljes felhasználói lista is letöltésre került. Ez azt jelenti, hogy rosszhiszemű felhasználó betekintést kap a felhasználók listájába. Ez lehetővé teszi, hogy betörjön az oldalára vagy kéretlen üzenetekkel halmozza el.
Az ilyen *Ajax*-os ellenőrzés egyik hátulütője a sebesség. Ahogy azt már korábban jeleztem, az *Ajax* aszinkron mivoltából adódik, sose tudható, mennyi időn belül kapunk választ. Az én esetemben a böngésző és a szerver közötti adatforgalomra szinte nem kellett várni. Egy terhelt szerver, egy összetettebb adatbázis lekérdezés vagy lassú internet kapcsolat esetén azonban már lomhának érezhetjük az aszinkron hívásokat. Azonban még a legrosszabb *Ajax* függvény is gyorsabb, mint a teljes oldal újratöltése, hiszen kevesebb az adatátvitel.

Zárszó

Ebben a részben végre elkezdtük használni az *Ajax*-ot az alkalmazásukban. Láttuk, miként kell egy meglévő *JavaScript* programot két függvényre bontani: az egyik az *Ajax* hívásért felel, a másik pedig feldolgozza a választ.
Természetesen láttuk a megoldás biztonsági és hatékonysági korlátait is. Jobb megoldás elküldeni csupán a felhasználói nevet és egy egyszerű igen/nem választ várni a szervertől, hogy foglalt-e már a felhasználói név. A következő hónapban a mostani *GET* helyett *POST* kérést fogunk küldeni és a statikus *usernames.txt* lecseréljük szerveroldali alkalmazásra, amely az *Ajax* hívásunkkal fog együttműködni.

Ajánlott olvasmányok

Az utóbbi időszakban robbanásszerűen bővült az *Ajax*-os irodalom, nehezen tudok velük lépést tartani. Két nagyon jó könyv van a témában. Az egyik a *Head Rush Ajax*. Ez első sorban a kezdőket célozza meg és hatékony, szórakoztató módon vezet be a rejtelmekbe. A másik a már korábban említett *Ajax Design Patterns*, amely minden bizonnyal a kedvenc *Ajax*-os könyvem (a kinézete és a felépítése ellenére, amely nem követi a szokásos *O'Reilly* hagyományokat). Ezutóbbi bevezetőnek ajánlott a gyakorlott webfejlesztőknek.
Az *Ajaxian.com* weboldal rengeteg linket, tananyagot és cikket tartalmaz *Ajax* fejlesztés témában többféle platformra. Érdemes felvenni az oldalt a *Kedvencek* közé vagy az *RSS* olvasónkba.

Linux Journal 2006., 151. szám

Reuven M. Lerner

Jelentések a weboldalon MySQL adatbázisból, CSS és Perl segítségével

A Maypole Futballklub alkalmazás kiegészítése egy egyszerű jelentéskészítő módszerrel.

A *Linux Journal 2005 márciusi* számában megjelent cikkben a *Maypole* rendszer használatával készítettünk webes adatbázis alkalmazást mindössze 18 sor *Perl* kód megírásával. A *Maypole* nyújtotta funkcionalitás egészen elképesztő egy fontos területet leszámítva, ez pedig a jelentések készítése. Így hát nekifogtam olyan technológiák kereséséhez, amelyekkel jelentéseket tudok készíteni a Futballklub alkalmazásból kinyert adatokkal. A célom az volt, hogy létrehozzak egy garnitúra gyakran használt jelentést, amelyeket a webes felületről lehet lekérdezni.

Webes jelentések? Mi a teendő?

Webes jelentések számtalan módon készíthetők a szokásos kiszolgáló oldali programnyelvek használatával, mint például a *PHP*, *JSP*, *Perl* parancsfájlok, és még sok más egyéb. Más, önálló munkaállomásokon futó jelentéskészítő eszközöket is használhatunk, amelyek még arra is képesek, hogy *MySQL* adatbázison alapuló jelentést készítsünk *OpenOffice.org* segítségével. Minthogy a jelentéskészítéssel kapcsolatosan csak alapvető elvárásaim vannak, így a szellemi ráfordítást is próbáltam minimalizálni. Azt nem bánom, ha a jelentést előállító *SQL* lekérdezés reszelgetésével kell tölteni az időt, de ha már meg van írva, azt szeretném, ha egyből egy *HTML* táblát adna eredményül.
Ezt persze megoldhatjuk *Perl*-ben, a *DBI* és *DBD::mysql* modulok segítségével, a kód kézi hegesztésével elküldött *SQL* lekérdezéseken keresztül. Ezt követően újabb kódok megírásával „utófeldolgozhatjuk”, mielőtt újabb kódrészletek felhasználásával végre elkészítenénk a táblázatot.
Az én egyszerű elvárásaimnak ez túl sok munkát jelentett. Amire tényleg szükségem volt, az egy gyors és igénytelen megoldás. A cikk hátralévő részében ezt a bizonyos általam készített megoldást fogom részletezni.

MySQL-t az adatok kinyeréséhez!

Miközben *Paul DuBouis* kiváló *MySQL* szakácskönyvét böngésztem, találtam egy parancssori kapcsolót, amely a szintén parancssorból átadott lekérdezés eredményét *HTML* táblázattá alakítja át (1.23-as recept, 33. oldal). Példa gyanánt nézzük a következő parancsot:

```
mysql  -e  "select name from
➥player"              \
    -u  manager  -ppwhere  CLUB
```

Ez a következő szöveges kimenetet eredményezi:

```
+-------------+
|name         |
+-------------+
|Robert Plant |
|Tim Finn     |
|James Taylor |
|Bryan Adams  |
|Ian Gillen   |
|Mick Jagger  |
|Neil Young   |
|Bob Dylan    |
+-------------+
```

Az eredményből a Futballklub összes játékosának neve kiderül, s úgy tűnik, hogy a klub játékosait híres folk és rock énekesek után nevezték el.
Ha a fenti parancsot újrafuttatjuk a *HTML* készítő kapcsolót alkalmazva,

```
mysql  -H -e  "select name from
➥player"              \
    -u  manager  -ppwhere  CLUB
```

akkor a következő szöveghalmazt kapjuk, ami – higgyék el nekem – egy *HTML* táblázat:

```
<TABLE BORDER=1><TR><TH>name
➥</TH></TR><TR><TD>
Robert Plant</TD></TR><TR><TD>
➥Tim Finn</TD></TR>
<TR><TD>James Taylor</TD></TR>
➥<TR><TD>Bryan Adams
</TD></TR><TR><TD>Ian Gillen
➥</TD></TR><TR><TD>
Mick Jagger</TD></TR>
➥<TR><TD>Neil Young</TD></TR>
<TR><TD>Bob Dylan</TD>
➥</TR></TABLE>
```

Az *SQL* lekérdezést eltehetjük egy fájlba is, aztán később hivatkozhatunk erre a fájlra parancssorból. Az alábbi példa – amelyben feltételeztük, hogy a fenti lekérdezés a *name.sql* fájlban található – ugyanazt a *HTML* táblát adja eredményül:

```
mysql  -H  -u  manager
➥-ppwhere  CLUB  <  name.sql
```

Ennek ismeretében azt találtam ki, hogy ha a *HTML* táblázatot előállító parancsot webes felületen keresztül indítanám el, akkor a nehezén rögtön túl is volnék, már ami a webes jelentéskészítő megoldásommal kapcsola-

tos problémákat illeti. Így hát készítettem egy *Perl* nyelven írt kis *CGI* parancsfájlt, amely a parancssoros utasításokat indítja.

A CGI szkript

A *CGI* szkriptben használt stratégia lényegre törő: miután megállapítottuk, hogy mi a neve a futtatandó lekérdezésnek, egy parancsot rakunk össze, majd elindítjuk azt a programból. Ezt követően a parancs visszatérési érékét a *CGI* által létrehozott *HTML* fájl body részébe illesztjük. A szokásos *Perl* indítósorokat leszámítva a ***runquery.cgi*** parancsfájl egy sor konstans meghatározásával kezdődik:

```
#! /usr/bin/perl -w
use strict;
use constant MYSQL  => '/usr/
➥bin/mysql';
use constant USERID =>
➥'manager';
use constant PASSWD =>
➥'pwhere';
use constant DBNAME => 'CLUB';
```

Elképzelhető, hogy a *MySQL* ügyfélprogram helye máshol van, mint az én számítógépemen, ebben az esetben javítsuk a konstans értékét. Fontos még, hogy beledrótoztam a felhasználó (`USERID`), jelszó (`PASSWD`) és adatbázis (`DBNAME`) értékeket a kódba. Ez ugyan nem a legszebb megoldás, de szeretném kimagyarázni magamat: ez az igénytelen része a bevezetőben említett gyors és igénytelen megoldásomnak. A megadott konstansokból már látszik, hogy a *Perl CGI* programnyelv szabványos felületét fogjuk használni:

```
use CGI qw( :standard );
```

Két *Perl* változót definiálunk, amelyek a webes felületről a parancsfájlnak átadott paraméterek értékét veszik fel. Az első paraméter neve `query`, ez határozza meg, hogy melyik *sql* fájlt fogjuk használni. A másik neve `title`, ebben van a jelentés címe amit az eredmények megjelenítésénél használunk.

```
my $query = param( 'query' );
my $title = param( 'title' );
```

A szkript ezután elkészíti a parancsot, amely lefuttatja a megadott lekérdezést a *MySQL* kliensen keresztül. Ne feledjük, hogy a *Perl*-ben a ponttal jelölt művelet a karakterláncok összefűzését jelenti.

```
my $cmdline = MYSQL .
              ' -H -u ' .
              USERID .
              ' -p' .
              PASSWD .
              ' ' .
              DBNAME .
              "< $query ";
```

Ezek után felépítjük a *HTML* oldalt. A `header` (fejléc) függvény elkészíti a helyes `Content-Type` fejléceket, aztán a `start_html` függvény létrehozza a *HTML* oldalt, amelynek címe a paraméterként átadott oldalcím lesz:

```
print header;
print start_html( -title =>
➥$title );
```

A következő kódsor a *Perl* `qx` műveletét használja a parancs elindításához, s a parancs kimeneti értékével tér vissza, amelyet a `$result` nevű változóba teszünk.

```
my $results = qx/ $cmdline /;
```

A parancsfájl további része egy 3. szintű címsort tesz a weboldalra, a lekérdezés eredményével együtt, majd egy *HTML* hivatkozást, amely visszamutat a jelentések oldalra. Az `end_html` függvény lezárja a *HTML* oldal generálását, és befejezi a szkript futását.

```
print "<h3>$title</h3>";
print $results;
print p, "Return to the ",
      a( { -href => "/Club/
➥Reports.html" },
      "List of Reports" );
print end_html;
```

A parancsfájl hívása

A szkript futtatásához két dolgot kell tennünk: el kell helyezni a parancsfájlt olyan helyre, ahonnan a webkiszolgáló eléri, valamint létre kell hozni az *SQL* lekérdezést tartalmazó fájlt. Az általam használt *Fedora Core 3* terjesztés *Apache 2* webkiszolgálót futtat, és a */var/www/cgi-bin/* könyvtárban tárolja a *CGI* parancsfájlokat. Egyszerűen csak át kell másolni a *CGI* fájlt erre a helyre, majd futtathatóvá kell tenni:

```
cp  runquery.cgi  /var/www/
➥cgi-bin/
chmod  +x  /var/www/cgi-bin/
➥runquery.cgi
```

A fenti elérési út lehet, hogy nem ugyanaz, mint amit az olvasó rendszere is használ, legelőször ennek járjunk utána. Lekérdezés gyanánt pedig használjuk az alábbit, amelyet a ***conditions.sql*** fájlba mentsünk:

```
select player.name as 'Player',
       condition.name as
➥'Medical Condition'
from   player, condition
where  player.medical_condition
➥= condition.id and
       player.medical_condition
➥!= 1;
```

Ez a lekérdezés összekapcsolja a `player` (játékos) és `condition` (erőnlét) táblákat, hogy ki tudja listázni az egyes játékosokat és a hozzájuk tartozó egészségügyi erőnlétre vonatkozó információt – feltételezve, hogy mindenkihez csak egy ilyen adat tartozik. A lekérdezést tartalmazó fájlt még át kell másolnunk a webkiszolgáló *CGI* könyvtárába:

```
cp conditions.sql /var/www/
➥cgi-bin
```

A lekérdezés *CGI* szkripten keresztül történő indításához gépeljük be a böngészőnk címsorába a következőt (a `localhost` tagot helyettesítsük a webkiszolgálónk tartománynevével):

```
http://localhost/cgi-bin/
➥runquery.cgi?           \
     title=Results&query=
➥conditions.sql
```

Az *URL* az ***1. ábrán*** látható eredményt állítja elő, amely a kevés előkészület ellenére is teljesen jónak tűnik, igaz lehetne szebb is.

Tegyük szebbé a dolgokat CSS használatával

Ahhoz, hogy egy tökéletesebb kinézetű jelentéshez jussak, készítettem egy kis *CSS* (*Cascading Style Sheet*) fájlt,

amelyet ***reports.css***-nek neveztem. Ez majd rendbe hozza a jelentésünk általános kinézetét.

```
body {
    font-family:    sans-serif;
}
table {
    font-family:    sans-serif;
    background-color:
➥LIGHTYELLOW;
}
table th {
    background-color:
➥LIGHTCYAN;
    font-size:      75%;
}
h3 {
    font-family:    sans-serif;
    color:          BLUE;
}
```

A stíluslapok működéséből adódóan a fájl meglehetősen egyszerű. Először is meghatározzuk az oldal alap betűtípusát, aztán trükközünk kicsit a jelentést alkotó táblázat hátterével és betűtípusával. A táblázatok fejléce 75%-a az alapértelmezett betűméretnek, és a táblázat adatainak háttérszíne is eltérő. Aztán megadjuk, hogy a jelentésben használt 3-as szintű címsor kék színű legyen.
A *CSS* fájlt a webkiszolgáló gyökérmappájába kell másolni, ahol a webalkalmazások is láthatják:

```
cp reports.css /var/www/html
```

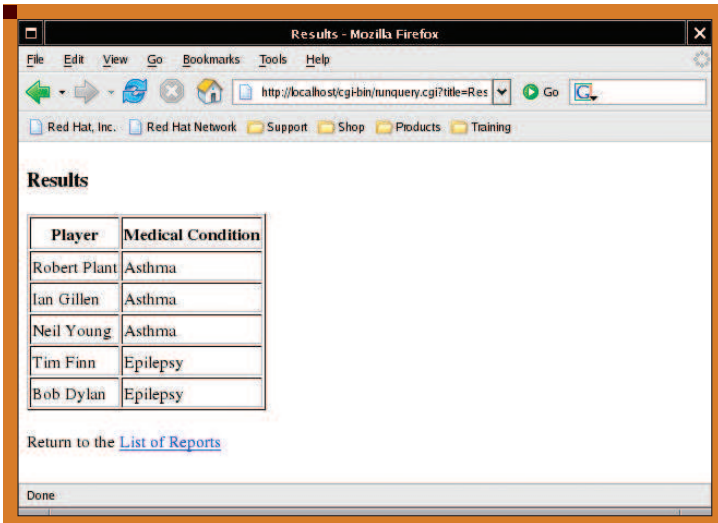
A *CSS* fájl használatához meg kell változtatnunk a ***runquery.html*** fájlban található `start_html` függvény paraméterezését, hogy az hivatkozzon a stíluslapra:

```
print start_html( -title =>
➥$title,
                  -style => {
➥-src => "/reports.css" } );
```

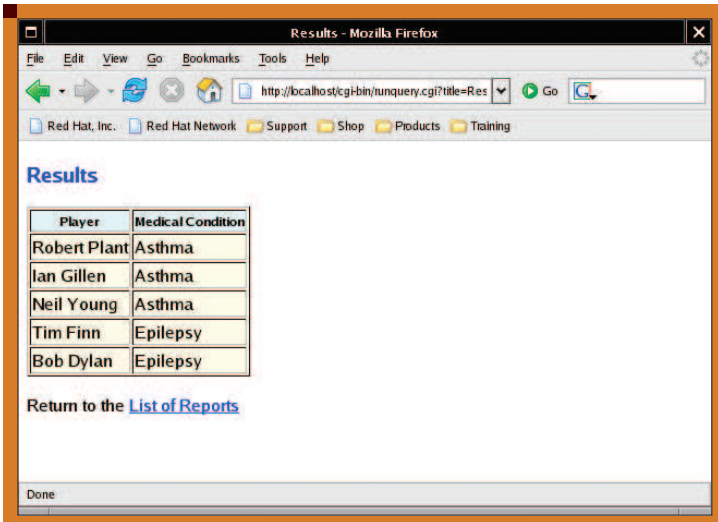
A *CGI* szkriptet újratöltve a *2. ábrán* látható eredményt kapjuk. Talán nem fog webdizájn díjat nyerni, de sokkal jobban néz ki, mint az *1. ábrán* látható egyszerű kimenet.

A webes felület elkészítése

A megoldásnak ez a része igen egyszerű. Egy sima weblapra van szükségem, amely a jelentések listáját tartalmazza. Hasonlóan, mint az előállított jelentések esetében, az előbbi nem túl bonyolult stíluslapot fogom használni a kinézet szebbé tételéhez. Íme a ***HTML***, amelyet készítettem:

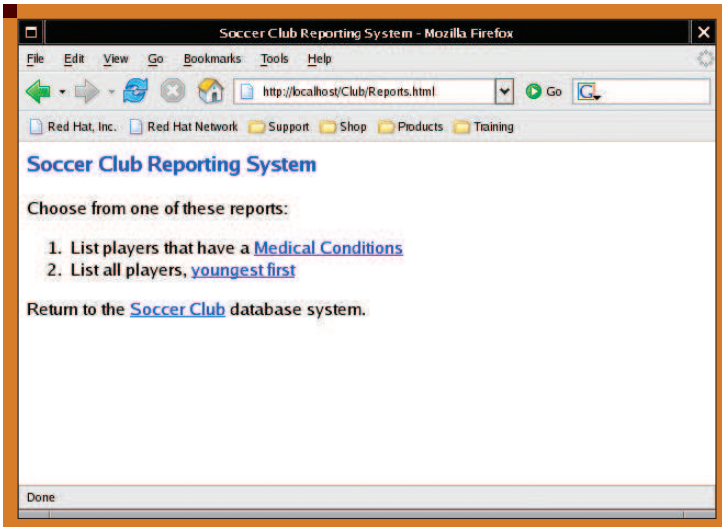


1. ábra Egy működőképes, bár nem túl szép HTML jelentés



2. ábra Egy sokkal tökéletesebb HTML jelentés

```
<HTML>
<HEAD>
    <TITLE>Soccer Club
➥Reporting System</TITLE>
    <LINK rel="stylesheet"
➥type="text/css"
          href="/reports.css" />
</HEAD>
<BODY>
<H3>Soccer Club Reporting
➥System</H3>
Choose from one of these
➥reports:
<OL>
    <LI>Mutasd azokat a játé-
➥kosokat, akiknek elérhető az
    <a href="/cgi-bin/
➥runquery.cgi?
    title=Players with a
➥Medical Condition&
    query=conditions.sql">
➥erőnléti eredménye</a>
    <LI>Mutasd az összes
➥játékost,
    <a href="/cgi-bin/
➥runquery.cgi?
    title=Listing of all
➥Players (Youngest First)&
    query=desc_dob.sql">kezdd a
➥fiatalokkal</a>
```



3. ábra A Jelentéskészítő felülete

```
</OL>
Vissza <A HREF="/Club">Soccer
➥Club</A>
az adatbázishoz.
</BODY>
</HTML>
```

Ahogy már láttuk, minden jelentést két paraméterrel hívunk meg: az egyik a `title`, amely a jelentés címét tartalmazza, a másik a `query`, amely azonosítja a lekérdezést, amelyet a *MySQL* ügyfélprogram segítségével le kell futtatni. A weblapot az elkészítés után másoljuk a Futballklub weboldalunk gyökérmappájába:

```
cp Reports.html /var/vvv/html/
➥Club
```

A böngészőből betöltve a jelentéskészítő webes felület a *3. ábrán* látható formában jelenik meg.
Ezen a ponton azt hiszem, készen vagyunk. Van egy egyszerű webfelületünk egy alap jelentéskészítő mechanizmushoz. Ha később több lekérdezést is írunk, elhelyezhetjük őket ***sql*** fájlokban egyenként, majd bemásolhatjuk a ***cgi-bin*** könyvtárunkban, frissíthetjük a Jelentéskészítő *HTML* kódját, hogy el tudjuk indítani ezeket a lekérdezéseket. A megoldás gyors és igénytelen, de több, mint elegendő.

Vagy mégsem?

A megoldásom biztonsági vonatkozását tekintve igen gyenge. Két problémára kell megoldást találnom: meg kell védenem a CGI és SQL parancsfájlokat attól, hogy a felhasználók trükközhessenek vele, valamint meg kell védenem a rendszert a CGI parancsfájloktól

Biztonság: Védelem a felhasználói belebabrálásoktól

Bár a *CGI* és *SQL* fájlokkal kapcsolatos felhasználói piszkálódásokról beszélünk, tudni kell, hogy a probléma minden olyan fájl esetében megvan, amelyek olvashatók a webkiszolgálón héj fiókkal rendelkező felhasználók számára. Egy sima `cat` vagy `less` paranccsal megnézhetjük a tartalmát. Minden felhasználó belenézhet a ***runquery.cgi*** fájlba, és kiolvashatja az adatbázishoz tartozó felhasználónév - jelszó párost , ami nem szerencsés.
Az ***User*** és ***Group*** tulajdonságok az ***Apache httpd.conf*** beállítási fájljában meghatározzák, hogy melyik felhasználó és csoport nevében fut a webkiszolgáló. A saját rendszeremen ez a felhasználónév és csoportnév az `apache`. Ennek ismeretében adjunk ki olyan parancsot, amely beállítja, hogy a *CGI* és *SQL* fájljaink az `apache` felhasználó legyen a tulajdonosa, valamint csak ez a bizonyos `apache` felhasználó tudja ezeket a fájlokat írni és olvasni. Ez a rendszergazda (root) felhasználó kivételével mindenki számára megakadályozza, hogy hozzáférjen a fájlok tartalmához.

```
cd  /var/www/cgi-bin
chown  apache:apache  *
chmod  600  *
chmod  700  *.cgi
```

Az első `chmod` utasítás hatására a ***cgi-bin*** könyvtárban található tartalom kizárólag az `apache` felhasználó számára lesz olvasható. A második `chmod` utasítás az összes *CGI* fájlra bebillenti a futtatható bitet, de csak a fájl tulajdonosa számára. Ezzel az egyszerű elővigyázatossággal a jelentéskezelőnk biztonságban van az illetéktelen felhasználói hozzáférésektől.

Biztonság: Védelem a CGI parancsfájlokkal szemben

A fenti `chmod` utasítások megvédik a fájlokat azoktól a felhasználóktól, akiknek héj fiókjuk van a kiszolgálón, de a jelentéskezelőnk még mindig sebezhető. Sajnos ezt minden felhasználó kihasználhatja egy egyszerű webböngésző segítségével. Tekintsük például, hogy mi történik, ha az alábbi *URL* használatával futtatjuk a *CGI* parancsfájlt:

```
http://localhost/cgi-bin/
➥runquery.cgi?            \
title=Ha!&query=conditions.sql
➥| cat runquery.cgi
```

A *CGI* parancsfájl tartalma megjelenik a böngészőnkben, és a támadó könnyedén kiolvashatja az adatbázis nevét, valamint a felhasználónév illetve jelszó párost. Ez már önmagában elég baj, de képzeljük el mi történik, ha a fenti *URL*-ben szereplő `cat runquery.cgi` szakaszt erre cseréljük:

```
cat /etc/passwd
```

vagy egy katasztrófával fenyegető változatra:

```
rm -rf /
```

Az általunk írt *CGI* szkripttel az a baj, hogy vakon megbízik a használójában, hogy az nem fog trükközni az *URL*-lel. Egy egyszerű csővezeték (pipe) karakter, és egy bármilyen héj parancs hozzáírásával a felhasználó kihasználhatja az átgondolatlanul megírt *CGI*-ből eredő biztonsági hibát: szó szerint bármilyen parancsot futtathat a kiszolgálón. A paraméterlista változatlan formában történő átadása az operációs rendszer számára nagyon sebezhetővé teszi a *CGI* parancsfájlokat.

Hála az égnek, a *Perl*-nek van egy különleges működési módja, amely segíthet nekünk. Ennek a neve ***taint*** (fertőzés) mód. A legtöbb *Perl*-lel foglalkozó könyv ismerteti a ***taint*** mód használatát, sőt, ***Christiansen*** és ***Torkington Perl*** szakácskönyve egy kényelmes megoldást is bemutat (19.4 recept, 767. oldal). A ***taint*** mód bekapcsolásával arra utasítjuk a *Perl* értelmezőt, hogy ne bízzon meg abban az adatban, amely a parancsfájlon kívülről származik. Mivel az adat megbízhatatlan, fertőzöttnek tekinti, a *Perl* a nem biztonságos változó felhasználása során egy futásidejű kivételt generál.
A ***Perl taint*** módja a *CGI* parancsfájlunk legelső során megváltoztatásával, a ***taint*** mód kapcsoló hozzáadásával érhető el.

```
#! /usr/bin/perl -wT
```

Ha újratöltjük a trükkös ***URL***-t, az eredményoldal üres lesz, és az ***Apache*** hibanaplója is gazdagodik egy 'nem biztonságos függőségi hiba' bejegyzéssel. A *Perl* így adja tudtunkra, hogy a szkript futása meghiúsult fertőzéses hiba okán. A futás meghiúsulása kétségtelenül megszünteti a rendszer biztonsági kockázatát, bár hozzá kell tenni, hogy a szkript így már nem látja el azt a feladatát, amire terveztük, azaz a megoldás minden, csak épp nem használható. Hogy újra életet leheljünk bele, meg kell tisztítanunk a bemeneti adatokat a *Perl* szabályos kifejezések használatával. Az ötlet igen egyszerű: meghatározunk egy a biztonságos adatnak megfelelő mintát, ezt a mintát illesztjük aztán a bejövő adatra, s abban az esetben, ha a minta illeszkedik, megbízhatóként kezelhetjük tovább. A *CGI* parancsfájlunknak két bemeneti adata van: a `query` és a `title` nevű paraméterek. A következő szabályos kifejezés hozzáadásával ellenőrizhetjük a bemenetek megbízhatóságát.

```
$query =~ /^([-\w]+\.sql)$/;
$query = $1;
$title =~ /^([\w:.?! ]+)$/;
$title = $1;
```

Az első szabályos kifejezés arra a karaktersorozatra illeszkedik, amely betűket és kötőjeleket tartalmaz tetszőleges elrendezésben, ponttal a végén, majd s q és l betűkkel zárva. Minden más, nem illeszkedő karaktersorozatot gyanúsnak tekintünk. Ha illeszkedik a mintára, a *Perl* a `$1` változóban fogja tárolni, amit aztán visszaírunk a most már megbízható `$query` változóba.
A cím paraméter mintája minden betűt elfogad, ezen kívül azokat a karaktereket is, amelyeket a fenti kifejezésben szögletes zárójelek között találunk. A `$title` változó szintén megbízhatóvá válik, ha a *Perl* illeszkedést észlel a futás során.
Mivel a *CGI* külső futtatható parancsot indít, nevezetesen a ***MySQL*** ügyfélprogramot, a futtatókörnyezet elérési útját szintén megbízhatóvá kell nyilvánítani. Ezt a `PATH` változó megfelelő értékre történő állításával érhetjük el, amely a biztonságos mappákat fogja tartalmazni:

```
$ENV{'PATH'} = "/usr/bin";
```

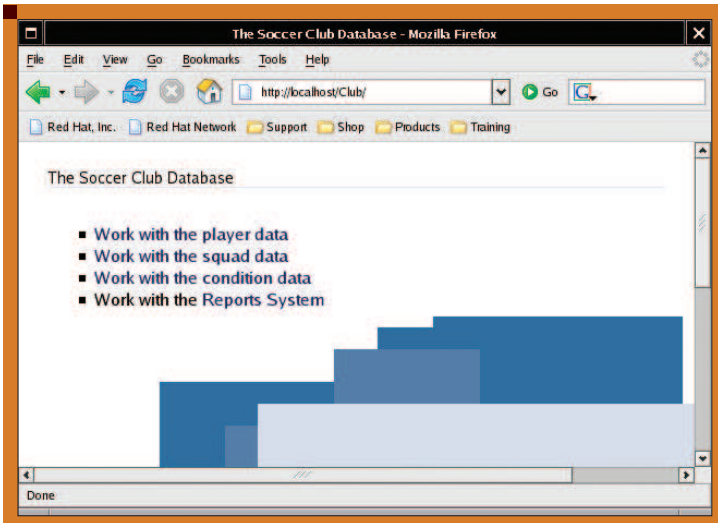
A fenti változtatásokkal biztonságossá tettük a ***runquery.cgi*** parancsfájlt, amely ismét használható. Ezen kívül egyszerű és igénytelen, és biztonságos a felhasználói támadásokkal szemben, a jelentéskezelő megoldásunk nem jelent többé potenciális veszélyt a rendszerünk számára.

A jelentéskészítő illesztése a Maypole alkalmazáshoz

Ahhoz, hogy hivatkozhassunk a Futballklub alkalmazásból a jelentéskezelőre, meg kell változtatni a ***custom/frontpage*** sablont, egy új listával bővítve, amely a jelentéskezelő oldalára mutat.

```
<ul>
[% FOR table =
➥config.display_tables %]
    <li>
        <a href="[%table%]/
➥list">Munka a(z)
                    [%table %]
➥tábla adataival</a>
    </li>
[% END %]
    <li>Munka a <a href=
➥"Reports.html">
          Jelentéskezelővel</a>
</ul>
```

Amikor betöltjük az alkalmazást a böngészőnkben, a hivatkozás a ***Maypole*** menü részeként jelenik meg, ahogy az a *4. ábrán* is látható. A webalapú jelentéskezelő rendszerünk egyszerű, biztonságos és könnyen bővíthető. Ehhez csupán annyi dolgunk van, hogy írjunk még néhány ***SQL*** lekérdezést.



4. ábra A Jelentéskezelő webfelületének beillesztése a Maypole alkalmazásba

Linux Journal 2006., 149. szám

Paul Barry (paul.barry@itcarlow.ie) előadó a Carlow-i Műszaki Egyetemen, Írországban. Az előadásainak anyagai, valamint az általa írt könyvek és cikkek megtalálhatók a weboldalán: glasnost.itcarlow.ie/~barryp.

Gyors alkalmazásfejlesztés GNOME alá Mono-val

Fejlesszünk GNOME környezetben a nyílt forrású .NET keretrendszerrel, a Mono-val.

A *Mono* a *.NET* fejlesztői környezet hatékony, nyílt forrású megvalósítása. Alkotóelemei egy közös nyelvi infrastruktúra (*CLI*, azaz *Common Language Infrastructure*) virtuális gép, egy *C#* fordító és számos osztálykönyvtár. *C#* nyelv és futtató környezet megvalósítása megfelel az *ECMA 334* és *335* szabványoknak.
A *Mono* – ami egyébként spanyolul majmot jelent – különböző osztálykönyvtárakat biztosít, többek között a *.NET* keretrendszer fejlesztői csomagjának egy nyílt forráskódú megvalósítását. Ebben a cikkben a *Mono* egyik leghasznosabb szolgáltatását vesszük górcső alá: a *GNOME* támogatását *Gtk#* formájában.
A *Gtk#* egy *.NET* nyelvi összekötő a *Gtk+* eszközkészlethez és számos más *GNOME* könyvtárhoz. Több, mint egyszerű csomagoló, hatékony platform grafikus felületű alkalmazások fejlesztéséhez *GNOME* környezetben. A *GTK#* nyelvi kötései kitűnő objektum-orientált alapot biztosítanak *C#* stílusú tervezéshez, egyszerűvé, mégis rugalmassá és hatékonnyá téve a *GNOME* fejlesztést.
A cikkben egy egyszerű *C#* alkalmazás szerkezetét tanulmányozzuk. A legegyszerűbb *„Hello, World!"* alkalmazással kezdjük, végül egy egyszerű *Wikipedia* keresőprogrammal fejezzük be. A függőségek mindössze a *Mono* és a *Gtk#* lesznek. Ezek a csomagok a legtöbb terjesztéshez elérhetők – lásd a megfelelő online forrásokat.

Hello, World!

Kezdjük a lehető legegyszerűbb *Mono* alkalmazással, mely a *„Hello, World!"* karakterláncot írja a képernyőre:

```
using System;
class first {
    public static void Main
    ➥(string[] args)
    {
        Console.WriteLine
        ➥("Hello, World!");
    }
}
```

Nyissuk meg kedvenc szerkesztőprogramunkat, másoljuk be ezt a kódot és mentsük *first.cs* néven. Ezután a következő paranccsal lefordíthatjuk a programot egy futtatható állománnyá:

```
$ mcs first.cs
```

Végül a következő paranccsal futtathatjuk:

```
$ mono first.exe
Hello, World!
```

Ez az alkalmazás tartalmazza az első osztályt. Minden programban szükség van egy belépési pontra, egy kezdőfüggvényre az osztályban, ahonnan a *Mono* futtatókörnyezet elkezdheti a program futtatását. Ez a függvény a Main, ahogyan a *C*-ben és a *C++*-ban is. A függvény prototípusa az alábbi:

```
public static void Main
➥(string[] args)
```

Programunk Main függvényében meghívunk egy egyszerű, *WriteLine* nevű függvényt, amely a `Console` osztályban található. A függvény a `printf()`-hez hasonlóan szöveget ír a kimenetre. Arra is használhatjuk, hogy változók értékét kiírassuk vele:

```
int x = 5;
String s = "wolf";
Console.WriteLine ("x={0}
➥s={1}", x, s);
```

Eredményül ezt kapjuk:

```
x=5 s=wolf
```

A Hello, World! színesben

A *„Hello, World!"*-öt természetesen nem csak a konzolon jeleníthetjük meg, *Gtk#*-pal egy egyszerű *GUI* párbeszédablakot is készíthetünk neki:

```
using System;
using Gtk;
class Two {
    static void WindowDelete
    ➥(object o,
    ➥DeleteEventArgs args)
    {
        Application.Quit ();
    }
    static void InitGui ()
    {
        Window w = new Window
        ➥("My Window");
        HBox h = new HBox ();
        h.BorderWidth = 6;
        h.Spacing = 6;
        w.Add (h);
        VBox v = new VBox ();
        v.Spacing = 6;
        h.PackStart (v, false,
        ➥false, 0);
        Label l = new Label
        ➥("Hello, World!");
        l.Xalign = 0;
        v.PackStart (l, true,
        ➥true, 0);
        w.DeleteEvent +=
        ➥WindowDelete;
        w.ShowAll ();
    }
    public static void Main
    ➥(string[] args)
    {
        Application.Init ();
        InitGui ();
```

```
        Application.Run ();
    }
}
```

Csakúgy, mint az előbb, írjuk be a kódot kedvenc szerkesztőnkbe, és mentsük *two.cs* néven. A program fordításához meg kell mondanunk a *Mono* fordítónak, hogy a *Gtk#* könyvtárat szeretnénk használni:

```
$ mcs two.cs -pkg:gtk-sharp
```

Futtatni ugyanúgy kell, mint az előbb:

```
$ mono two.exe
```

Az alkalmazás létrehoz egy kis ablakot, amelynek címe *My Window* lesz és a *„Hello, World!"* üzenetet jeleníti meg benne (*1. ábra*). Az ablak egy `GtkWindow`, a címke pedig egy `GtkLabel`. A *Gtk* az ablakokat dobozokra osztja. A dobozok láthatatlan grafikus elemek, kizárólag az elrendezés miatt léteznek, vagyis, hogy más elemeket tartalmazzanak. A elemek elrendezését az ablakon belül a elemek dobozbeli elrendezése határozza meg. Bár *Gtk*-ban az elemek táblázatokkal is elrendezhetők, a legtöbb programozó dobozokat használ azok rugalmassága és hatékonysága miatt. Ráadásul, ha egyszer alaposan megismerjük a dobozokat, utána már egyáltalán nem nehéz használni őket.
A *Gtk*-ban két doboztípus van: a függőleges és a vízszintes. A `vboxnak` nevezett függőleges doboz az elemek függőleges elrendezését határozza meg – oszlopokba rendezi őket. A `hbox` vízszintes doboz az elemek vízszintes elhelyezkedéséért felel, sorokba rendezi őket. Az elemeket dobozokba tesszük, azokat újabb dobozokba, amelyeket végül az ablakokhoz adunk.
Új `hbox`-ot így hozunk létre:

```
HBox h = new HBox ();
```

Új `vbox`-ot pedig így:

```
VBox v = new VBox ();
```

A dobozokat jelölő új objektumoknak különféle tulajdonságaik vannak, melyek beállításával a doboz kinézete (look and feel) szabályozható.
A következő példában beállítjuk a `hbox` két tulajdonságát:

```
h.BorderWidth = 6;
h.Spacing = 6;
```

Itt a `hbox` körüli szegélyt és térközt határoztuk meg, mindegyiket hat képpontra, így a létrejött rács térköze 12 képpont lesz. A *GNOME HIG* (*Human Interface Guideline*) esztétikai okokból és az egységesség miatt ehhez hasonlóan 12 képpont távolságot ír elő az elemek között – így a példában szereplő 6+6 képpont tökéletes.
Dobozt az ablakhoz úgy adhatunk, hogy az ablak `Add()` tagfüggvényének átadjuk a dobozt:

```
w.Add (h);
```

Elemet dobozhoz pedig a doboz `PackStart()` tagfüggvényével adhatunk:

```
public void PackStart (Widget
➥child,
            bool expand,
            bool fill,
            uint padding)
```

Példánkban ez így néz ki:

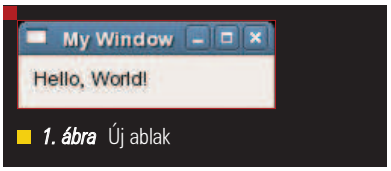
```
v.PackStart (l, true, true, 0);
```

Ez a függvényhívás a címkénket a `vbox`-ba teszi. Ha az `expand` (kiterjeszt) paraméter értéke `true`, a gyermekelem a doboz összes rendelkezésre álló területét kitölti. Ha a `fill` (kitölt) paraméter `true`, az elem a megjelenítésre felhasználja egész területét; ha `false`, akkor a felesleges területet üres lesz. A `padding` (kitöltés) paraméterrel az elem köré további térközt adhatunk a dobozon belül, más, eddig megadott térközön túl.
Alkalmazásunk futtathatóvá tétele három egyszerű lépésből áll:

```
Application.Init ();
InitGui ();
Application.Run ();
```

Az `Application.Init()` beállítja a *Gtk#*-ot és az alkalmazás grafikus felhasználói felületét. Ennek a *Gtk#* alkalmazások által meghívott első függvények között kell lennie. Ezután a program beállítja a *GUI*-ját, létrehozza és elrendezi a grafikus elemeket, megjeleníti a kezdő ablakokat és más *UI* elemeket. Ebben a programban ezt az `InitGui()` függvénnyel hajtjuk végre. Ha minden készen van, a program meghívja az `Application.Run()` függvényt, s indulhat a móka. Főablakunk felbukkan, mert az `Init.Gui()` tagfüggvényben erre utasítottuk:

```
w.ShowAll ();
```



1. ábra Új ablak

Ez megjeleníti az ablakot, benne az összes grafikus elemmel. Tehát, ha egyszer az alkalmazás meghívta az `Application.Run()`-t, felhasználói felületi elemeink megjelennek.
Futás közben a program válaszait az elemek viselkedése határozza meg. Vannak olyan elemek, amelyek előre meghatározott módon működnek, ilyenkor a programozónak nem kell saját kódot írnia. Gyakoribb azonban, hogy a programozó maga szeretné kezelni az eseményeket. Ehhez egy eseménykezelőt kell írnia, felhasználva a *C#* eseménykezelését, amelyen a *Gtk#* felhasználói beavatkozásra adott reakciói alapulnak.
Utolsó példánkban is egy ilyen eseménykezelő szerepel. Célunk az, hogy alkalmazásunk futása befejeződjön, amikor a felhasználó a főablak bezárás gombjára kattint. Kezelnünk kell tehát az ehhez kapcsolódó eseményt, a `DeleteEvent`-et, ehhez írunk egy eseménykezelőt:

```
static void WindowDelete
➥(object o, DeleteEventArgs
➥args)
{
    Application.Quit ();
}
```

Majd eseménykezelőként az ablakhoz adjuk:

```
w.DeleteEvent += WindowDelete;
```

Az `Application.Quit()` függvény hatására a *Gtk#* megsemmisíti a felhasználói felületet, bezárja és leállítja az alkalmazást. Következésképpen, ha a felhasználó rákattint a főablak bezárás gombjára, alkalmazásunk szépen befejeződik.

Egy jobb példa

Itt az ideje, hogy egy összetettebb, sőt, szinte már hasznos programocskát hozzunk létre: egy eszközt a *Wikipediában* való keresésre. Szögezzük le az elején, hogy semmi különlegeset nem csinálunk, de jól fogunk szórakozni. Létrehozunk egy

2. ábra A program grafikus felülete

egyszerű ablakot benne egy szövegbevitelre alkalmas elemmel. A felhasználó ide begépelheti a keresendő kifejezést, és egy gombra kattinthat (*2. ábra*). Az alkalmazás elindítja a felhasználó webböngészőjét és végrehajtja a keresést a ***Wikipediában***.
És ehhez csak néhány sor kódra van szükség, beleértve a ***GUI*** létrehozását és a böngészőbeli keresést is.
Az új programhoz a legutóbbi példát vesszük alapul, majd új elemeket és funkciókat adunk hozzá. A ***GTK#***-nak hála, nincs is sok új kódra szükség. Íme:

```
using System;
using Gtk;
class Example {
    public static Entry
    ➥search_entry;
    public static void
    ➥ButtonClicked (object o,
    ➥EventArgs args)
    {
        String s = "http://
➥en.wikipedia.org/wiki/Special:
➥Search?search=";
        s += search_entry.Text;
        s += "&go=Go";
        Gnome.Url.Show (s);
    }
    static void WindowDelete
➥(object o, DeleteEventArgs args)
    {
        Application.Quit ();
    }
    static void InitGui ()
    {
        Window w = new Window
➥("Wikipedia Search");
        HBox h = new HBox ();
        h.BorderWidth = 6;
        h.Spacing = 6;
        w.Add (h);
        VBox v = new VBox ();
        v.Spacing = 6;
        h.PackStart (v, false,
➥false, 0);
        Label l = new Label
➥("_Search:");
        l.Xalign = 0;
        v.PackStart (l, true,
➥false, 0);
        v = new VBox ();
        v.Spacing = 6;
        h.PackStart (v, true,
➥true, 0);
        search_entry = new Entry
➥();
        search_entry.Activates
➥Default = true;
        l.MnemonicWidget =
➥search_entry;
        v.PackStart
➥(search_entry, true, true,
➥0);
        v = new VBox ();
        v.Spacing = 6;
        h.PackStart (v, true,
➥true, 0);
        Button b = new Button
➥("Search");
        b.CanDefault = true;
        w.Default = b;
        v.PackStart (b, true,
➥true, 0);
        b.Clicked +=
➥ButtonClicked;
        w.DeleteEvent +=
➥WindowDelete;
        w.ShowAll ();
    }
    public static void Main
➥(string[] args)
    {
        Application.Init ();
        InitGui ();
        Application.Run ();
    }
}
```

Meg kell adnunk egy új, ***gnome-sharp*** nevű szerelvényt a fordítónk parancssorában. Ha programunkat ***three.cs***-nek neveztük el, ezt a következő módon tehetjük meg:

```
$ mcs three.cs -pkg:gtk-sharp
➥-pkg:gnome-sharp
```

És így futtatjuk:

```
$ mono three.exe
```

E harmadik és egyben utolsó programunk tartalmaz néhány új elemet – gombokat, címkéket és szövegmezőket –, de ugyanolyan egyszerű szerkezetű, mint az előző két példa. Ha visszafelé dolgozunk az utolsó dobozból kifelé az elsőig, már meg is van az elemek elrendezése anélkül, hogy akár egy képernyőképet is láttunk volna.
A másik nagy különbség egy új esemény, a `Clicked`. Meghatározunk egy függvényt és átadjuk eseménykezelőként:

```
b.Clicked += ButtonClicked;
```

Létrehoztuk egy nyilvános szövegbeviteli mezőt (***search_entry***), így, amikor a felhasználó megnyomja a gombot, az új eseménykezelőnk átveheti ennek a tartalmát a `search_entry.Text`-ből.
A `ButtonClicked` eseményben átvesszük a keresési kifejezést, létrehozzuk a kereséshez szükséges ***Wikipedia URL***-t, és a `Gnome.Url.Show()` függvénnyel megnyitjuk a felhasználó alapértelmezett webböngészőjét (ez egy globális ***GNOME*** beállítás), majd megnyitjuk vele az ***URL***-t.

Végszó

Nagyszerű, hogy ennyi mindent megvalósíthatunk ilyen kevés kóddal, az egészben a legszebb pedig az, hogy mindezt egy ***C*** alapú, s nem valamilyen parancsnyelven. A ***C#*** megőrzi a ***C*** hatékonyságát és teljesítményét, és ezzel együtt hatékony objektum-orientált programozási szerkezeteket is biztosít. Hadd legyek őszinte. ***C*** programozó vagyok, napjaim nagy részét kernelfejlesztéssel töltöm. A magasabb szintű nyelvek nem éppen a kedvenceim. Nem rajongok a ***C++***-ért és a ***Java***-ért sem. Mégis, mostanában, a ***C#***-pal dolgozom – például a Beagle keresőn – és teljesen lenyűgöz. A ***C#*** egy elegáns, jól megtervezett nyelv. ***Mono***-ban található szabad és nyílt ***C#*** fordító, futtatókörnyezet és számtalan könyvtár, és ráadásul élénk nyílt forrású közösség dolgozik rajta. Kitűnően alkalmas ***GNOME*** fejlesztésre, de sok más területen is jól alkalmazható.

Linux Journal 2006., 143. szám

Robert Love a Novell Ximian Desktop csoport vezető kernelfejlesztője, valamint a Linux Kernel Development (SAMS 2005) c. könyv szerzője. (A könyv második kiadása is megjelent már.) A Floridai Egyetemen szerzett diplomát CS-ből (computer science) és matematikából.

A CIKK FORRÁSAI

➲ www.linuxjournal.com/article/8750

Beágyazott Java GCJ-vel

Nincs feltétlenül szükség Java virtuális gépre ahhoz, hogy beágyazott rendszereken Java programot futtassunk.

Az alábbiakban bemutatjuk, hogyan kell használni a *GCC* fordítóprogramcsomag részét képező *GCJ*-t beágyazott rendszerekhez *Linuxon*. Mint minden eszköznek, a *GCJ*-nek is van olyan előnyös tulajdonsága, mely egyben a hátránya is, nevezetesen, hogy egy magas szintű nyelven, *Java*-ban programozhatunk. A *GCJ* beágyazott rendszereken való futtatásának ötlete először ijesztő lehet, de a cikk végére kiderül, hogy egyszerűbb mint gondolnánk. Reményeim szerint a cikk végére az Olvasó kellően felcsigázott lesz, hogy kipróbálja a tanultakat, és meggyőződjön róla, hogy érdemes a *GCJ*-re gondolnia a legközelebbi munkája során. A *Java* nyelv remek eszközkészlettel rendelkezik ahhoz, hogy gyorsan fejleszthessünk megbízható szoftvereket. Ennek része pl. az automatikus memóriaszemét-gyűjtés, a sokrétű és robusztus futásidejű könyvtár és a rendkívül kifejező objektum-orientált szerkezetek.

Miért pont a GCJ?

A natív *Java* fordító pontosan azt csinálja, amit a neve sugall: a *Java* forráskódot az adott hardver gépi kódjára alakítja át. Ebből az is következik, hogy az adott gépen nincs szükség *Java* virtuális gépre (*Java Virtual Machine – JVM*), a program futtatásakor nem töltődik be *JVM*, pontosan úgy töltődik be és fut le, ahogy bármely más végrehajtható program. Ez sajnos nem szükségszerűen jelenti azt is, hogy a program gyorsabban fut majd. Előfordulhat, hogy a *GCJ* által fordított kódnál jobb eredményeket kapunk egy natív virtuális gépen futó bájtkóddal.
A *GCJ* egyik előnye az, hogy mivel nincs szükség *JVM*-re, helyet takarítunk meg, továbbá pénzt takaríthatunk meg a jogdíjak miatt is. Mindezen túl, a *GCJ*-vel kizárólag nyílt forráskódú szoftverekkel állíthatunk elő új terméket, ami általában nyerő dolog.

Buktatók

Mikor a beágyazott rendszerekkel foglalkozó mérnökök gyökérfájlrendszert hoznak létre az adott platformon, az első dolguk az *uClibc*, egy kompakt *glibc* függvénykönyvtár telepítése. Azok kedvéért, akik nem járatosak a *Linux* beágyazott rendszereken való használatában, a standard *C* függvénykönyvtár óriásnak számít egy olyan környezetben, ahol a gyökérfájlrendszer pl. Legfeljebb 8 MB lehet. Helytakarékosság okán, a fejlesztők a standard *C* függvénykönyvtárat egy kisebbel váltják fel, pl. az *uClibc*-vel. Mivel a *GCJ* megköveteli az *Unicode* támogatását, a *glibc* nélkülözhetetlen, hiszen az *uClibc*-ben ilyen nincs.
A standard függvénykönyvtár a *GCJ*-nek 16 MB-ba kerül, így még ha tudnánk is kisebb standard *C* könyvtárra váltani, nagy különbséget nem jelentene. A standard *GCJ* könyvtárból eltávolíthatnánk a *Java* bájtkódok végrehajtásának támogatását, de összességében a *GCJ* hasznossága ettől jelentősen csökkenne.

A hoszt és a célplatform beállítása

Minthogy e cikk a *GCJ* beágyazott rendszereken való használatával foglalkozik, azt is bemutatja, hogyan készítsünk keresztfordítást és egyszerű gyökérfájlrendszert a célplatformon. Az újoncok kedvéért, a keresztfordító olyan végrehajtható kódot állít elő, amely a fordítást végző géptől különböző célplatformon futni képes.
A fordítót futtató gépet hosztnak, a kódot futtató gépet pedig célplatformnak nevezzük.
Esetünkben a célplatform egy 350 MHz-en ketyegő *PPC 745/755* alapú gép. Ezt az alaplapot áttetsző dobozban árulják, merevlemezzel és kijelzővel, és *iMac* néven is szokás emlegetni. Belátom, ez nem a legjobb példa egy beágyazott rendszerre, de ugyanolyan problémákkal foglalkozik, amelyekkel a valódi beágyazott rendszereken is szembesülünk. Az itt elsajátított ismeretek jól hasznosíthatók lesznek más processzoroknál is.
A hoszt egy teljesen átlagos *IBM ThinkPad* noteszgép lesz, *Pentium III*-as processzorral. A gépen *Yellow Dog Linux* fut, amit majd kicsit módosítunk, hogy a cikkben készített gyökérfájlrendszert használja.

Helyezzük üzembe a GCJ-t

Először is, szükségünk lesz egy olyan keresztfordítóra, amely egyrészt futtatható a *Pentium III*-as gépen, másrészt képes a *PowerPC 750* processzoron futó kód előállítására. A keresztfordító készítése a célplatformra nagyon macerás lehet; egy működő *GCC* még nem feltétlenül jelenti azt, hogy használható fordítónk van. Néhány további eszközre is szükségünk lesz, például a *binutils*-ra és az adott nyelvhez tartozó standard függvénykönyvtárakra.
Ha gyorsan és fájdalommentesen akarunk keresztfordítóhoz jutni, használjuk a *crosstool* csomagot, *Dan Kegel* keze munkáját. A *crosstool* megcsinálja helyettünk az összes nehéz lépést, ami a keresztfordító készítéséhez szükséges: letölti a forráskódot és a javításokat, végrehajtja a javításokat, elvégzi a szükséges beállításokat a csomagon

és végül elkészíti a fordítót. A *crosstool* beszerzése és kicsomagolása után a következő lépéseket hajtsuk végre a *GCJ* keresztfordító létrehozásához:

```
$ export TARBALLS_DIR=~/
➥crosstool-download
$ export RESULT_TOP=/opt/
➥crosstool
$ export GCC_LANGUAGES=
➥"c,c++,Java"
$ eval `cat powerpc-750.dat
➥gcc-4.0.1-glibc-2.2.2.dat'
➥sh.all -notest
```

Míg a fordító teszi a dolgát, vessünk egy közelebbi pillantást az imént kiadott utasításokra. A `TARBALLS_DIR` változó mondja meg, hová töltse a *crosstool* a fájlokat. Alapértelmezés szerint, a *crosstool* a program készítéséhez szükséges összes fájlt letölti.
A `RESULT_TOP` definiálja a keresztfordító telepítőkönyvtárát. Végül, a `GCC_LANGUAGES` változó állítja be, hogy mely nyelvi front-endek kerüljenek be a fordítóba. A *GCC* rengeteg nyelvet támogat, és minden egyes front-end fordítása jelentősen megnöveli a fordítási időt, ezért a `GCC_LANGUAGES` változó csak azokat tartalmazza, amelyekre szükségünk van.
A *Bash* parancsnyelvében járatlanok kedvéért, az utolsó sor a képernyőre írja a két *.dat* fájl tartalmát, majd végrehajtja az *all.sh* parancsfájlt a `--notest` kapcsolóval. Az egyszerűség érdekében, a *crosstool* eleve rendelkezik azokkal a konfigurációs fájlokkal, amik a célprocesszorhoz és a *gcc*/*glibc* kombinációhoz szükséges környezeti változókat tartalmazzák. Esetünkben a *crosstool gcc 4.0.1*-et készít *glibc 2.2.2*-vel, *PPC 750* processzorra. A *crosstool* az összes ismert processzor és *glib*/*gcc* kombinációhoz rendelkezik konfigurációs fájlokkal. Az összeállítás után a keresztfordítót a *$RESULT_TOP*/*gcc-4.0.1-glibc-2.2.2*/*powerpc-750-linux-gnu*/*bin* könyvtárban találjuk. Legjobb, ha ezt az útvonalat rögtön a `PATH` környezeti változóhoz adjuk, hogy a fordító bárhol könnyen elindítható legyen.

A crosstool beszerzése és kicsomagolása

A *crosstoll Dan Kegel* munkájának gyümölcse. A ➲ *kegel.com/crosstool* címen mindent megtudhatunk a *crosstool*-ról, amit tudni szeretnénk. A teljes dokumentáció mellett egy kiváló gyorstalpalót is találunk. A cikk írásakor a ➲ *kegel.com/crosstool/crosstool-0.38.tar.gz* címen lévő 0.38-as verziót használtam.
A *crosstool* honlapján feltétlenül ellenőrizzük az összeállítások naplóját (➲ *kegel.com/crosstool/crosstool-0.38/buildlogs*), hogy tudjuk, a *glibc*/*gcc* milyen kombinációi fordulnak sikeresen a célplatformhoz.

A gyökérfájlrendszer beállítása

Újonnan készített keresztfordítónk első feladata a gyökérfájlrendszer fordítása lesz, amely ebben az esetben a *BusyBox* része. Azért, hogy a beavatatlanok is tudják, a *BusyBox* egy olyan, megdöbbentően kis méretű, végrehajtható bináris fájl, amely magában foglalja a legnépszerűbb *UNIX*-eszközöket, kifejezetten azoknak, akinek minden bájt számít. A *BusyBoxon* gombok százai állnak nyomásra készen, hogy létrehozhassunk kényszerű megkötéseink fájlrendszerét, a szükséges eszközök támogatásával. A példa kedvéért most keresztfordításhoz állítjuk be a *BusyBox* konfigurációját, a méretre történő optimalizálást pedig gyakorlatként az Olvasóra bízom.
A *BusyBox* a beágyazott *Linux* világának egyik fő tartópillére, melynek karbantartását *Erik Anderson* végzi. Letölthető a ➲ *www.busybox.net/downloads/busybox-1.01.tar.bz2* címről. A gyökérfájlrendszer létrehozásához adjuk ki a `make menuconfig` parancsot abban a könyvtárban, ahová a *BusyBoxot* kicsomagoltuk. Ez pontosan úgy működik, ahogy a 2.4/2.6 rendszermagok beállításának felhasználói felülete. Az alábbiakban bemutatom, mit kell tenni a gyökérfájlrendszer fordításához.
Először is, válasszuk ki az összeállításhoz szükséges kapcsolókat. Jelöljük ki a *„Keresztfordítóval szeretné összeállítani a BusyBoxot?”* (*Check the Do you want to build BusyBox with a Cross Compiler?*) feliratú jelölőnégyzetet. A felbukkanó szövegbeviteli mezőbe írjuk be a keresztfordító előtagját, ami esetünkben *powerpc-750-linux-gnu-*. A *BusyBox* parancsfájljai a fordítás közben ezzel fűzik össze a szükséges eszközök nevét (például `gcc` és `ld`). Győződjünk meg róla, hogy a fordító útvonala benne van a `PATH` környezeti változóban, és adjuk ki az alábbi parancsokat:

```
make
make install
```

Az új fájlrendszer a `./_install` könyvtárba kerül. Biztosan feltűnik majd, hogy a *BusyBox* lényegesen gyorsabban fordul, mint a *GCC*.

A gyökérfájlrendszer benépesítése függvénykönyvtárakkal

Már majdnem kész vagyunk, de az új fájlrendszerünk még teljesen üres, függvénykönyvtárak sehol. A *GCJ* programoknak szüksége van bizonyos könyvtárakra, így a *BusyBoxnak* is, ahogy ez az *1. Táblázatban* látható. Ezek pontosan azok a könyvtárak, amelyeket a keresztfordító is használ. Példánkban a fájlok a *$RESULT_TOP*/*gcc-4.0.1-glibc-2.2.2*/*powerpc-750-linux-gnu*/*powerpc-750-linux-gnu*/*lib* (figyelem, nem elírás!) könyvtárban találhatók, és egyszerűen átmásolhatjuk őket a gyökérfájlrendszerbe:

```
for f in ld.so.1 lib libdl.so.2
➥libgcc_s.so.1libgcj.so.6
 -->libm.so.6 libpthread.so.0 ;
➥do

cp
$RESULT_TOP/gcc-4.0.1-glibc-
➥2.2.2/powerpc-750-linux-gnu/
➥powerpc-750-linux-gnu/lib/$f
<busybox install directory>/lib

$RESULT_TOP/gcc-4.0.1-glibc-
➥2.2.2/powerpc-750-linux-gnu/
➥bin/power
pc-750-linux-gnu-strip <busybox
➥install directory>/lib/$f
done
```

A gyökérfájlrendszerben egy */proc* könyvtárat is kell készítenünk, a *proc* fájlrendszer becsatolási pontjának. Az éles szemű Olvasó bizonyára észrevette, hogy nem őriztem meg a könyvtárak különféle verzióinak kezelésére létrehozott szimbolikus linkeket – ez elterjedt gyakorlat a beágyazott rendszereknél, ahol a konfiguráció az eszköz élettartama során sohasem változik meg, eltérően az asztali gépektől. A `strip` futtatása jelentősen, mintegy 50 %-kal csökkenti a szükséges merevlemez-területet.

1. táblázat *A GCJ-hez és BusyBoxhoz szükséges könyvtárak*

Könyvtár	Leírás
ld.so.1	A programok futásához szükséges dinamikusan kapcsolt fájlok betöltéséről és kapcsolásáról gondoskodik.
libdl.so.2	A dinamikusan kapcsolt fájlok manipulálásához szükséges segédfüggvények gyűjteménye.
libgcc_s.so.1	A kivételkezeléshez szükséges programozói interfészt definiálja.
libgcj.so.6	A GCJ futásidejű könyvtára, amely a standard Java könyvtárak megvalósítását tartalmazza.
libm.so.6	Matematikai függvények könyvtára.
libpthread.so.0	POSIX-szálak függvénykönyvtára.

A gyökérfájlrendszert most már átmásolhatjuk a célplatformon lévő ***bbox*** könyvtárba. A rendszert a `chroot` paranccsal vehetjük rá, hogy ezt használja gyökérfájlrendszerként. Rendszergazdaként indított parancssorból adjuk ki a következő utasítást:

```
chroot /tmp/bbox /bin/ash
```

Ennek hatására a / becsatolási pont a */**tmp**/**busybox*** könyvtárra kerül, és elindul a */**bin**/**ash*** parancsértelmező. Működik? Gratulálok! Éppen most hoztunk létre egy gyökérfájlrendszert egy beágyazott rendszerhez, egészen az alapoktól. Nyugodtan veregesse meg a vállát mindenki!
GCJ-nek arra is szüksége van, hogy a ***proc*** fájlrendszert becsatoljuk a jelenlegi gyökérfájlrendszerbe, amit a chroot után a következő parancsok végrehajtásával oldhatunk meg:

```
mkdir /proc
mount -t proc none /proc
```

Igaz ugyan, hogy a mi gyökérfájlrendszerünk egy teljesen szokványos merevlemezen van, de nem sok különbséget találnánk, ha mindezt egy beágyazott rendszeren hajtottuk volna végre. Biztosan lenne két nélkülözhetetlen eltérés: egyrészt létre kell hoznunk az ***inittab***-ot, hogy az alaplap a megfelelő parancsfájlokat hajtsa végre induláskor, másrészt a */**dev*** fájlrendszert is létre kell hozni a célplatformnak megfelelő eszközökkel.

Fejlesztés GCJ-vel

A keresztfordító és a gyökérfájlrendszer létrehozása után, az első ***GCJ***-alkalmazás létrehozása hihetetlenül egyszerű lesz. A tradicionális helló világ programmal kezdünk:

```
Class hello {
 Static public void main(String
➥argc[]) {
   System.out.println("hello
➥from GCJ");
 }
}
```

A ***Java*** konvenciók szerint ezt az osztályt a ***hello.class*** fájlban találjuk meg. A fordítás így történik:

```
powerpc-750-linux-gnu-gcj
➥hello.class --main=hello -o
➥hello-java
```

Mire jó a `--main=hello`? Bármely osztály definiálhat metódust egy alkalmas belépési ponttal. A `--main=hello` azt mondja a fordítónak, hogy a `hello` osztályban a main metódust használja az összeszerkesztésnél. Ha elhagyjuk, hibát kapunk: nem létező hivatkozás a main metódusra (`undefined reference to main`), ami a kezdők számára zavaró lehet, hiszen a `hello` osztályban van main metódus.
Töltsük fel a fájlt a célplatformra, és futtassuk a `chroot`-tal indított parancssori értelmezőből. Ezt fogjuk látni:

```
# ./java-test
Hello from GCJ
```

Ettől a ponttól kezdve, a fejlesztés a megszokott módon halad tovább, azzal a kivétellel, hogy a natív ***javac*** fordító helyett a ***GCJ*** keresztfordítót használjuk.

Helytakarékosság

A bemutatott példa gyökérfájlrendszere több mint 20 MB. Mivel a beágyazott rendszerek gyakran használnak ***Flash***-memóriát, melynek fajlagos költsége lényegesen nagyobb, mint a merevlemezes rendszereké, a lehető legkisebb méretű gyökérfájlrendszer általában alapvető követelmény.
A gyökérfájlrendszer méretét legegyszerűbben úgy csökkenthetjük, hogy az alkalmazást statikusan szerkesztjük. Bár az ösztöneink azt súgják, ez pont az ellenkezőjét eredményezi, a ***libc***-nek az alkalmazásba kerülő másolata miatt. Azt se felejtsük el azonban, hogy a ***libgcj.so*** a teljes ***Java*** standard függvénykönyvtárat tartalmazza, aminek a legtöbb alkalmazás csupán töredékét használja. A statikus szerkesztés így kiváló módja a sosem használt kód kirostálásának.
No és a `strip`-ről se feledkezzünk meg, különben megnézhetjük, hány bájtnyi hibakeresési információ lesz belegyömöszölve a ***libgcj.so*** fájlba.

Zárszó

A cikkből remélhetőleg kiderült, hogy a beágyazott rendszerekre történő fejlesztés ***GCJ***-vel megbízhatóan elvégezhető a jelenleg is elérhető, nyílt forráskódú eszközökkel. Bár néhány trükköt nem árt ismerni, az is nyilvánvaló, hogy a gyökérfájlrendszer konfigurálása nem nevezhető éppen heroikus küzdelemnek; a lényeg, hogy azokból a könyvtárakból, amelyekre egyébként is szükségünk lenne, néhány dolog elhagyható. Láthattuk, hogy a gyökérfájlrendszer mérete jelentősen csökkenthető az alkalmazásunk statikus összeszerkesztésével. Röviden, a ***GCJ*** lehet a mi barátunk, ha erőforrások szűkében lévő rendszeren kell ***Java***-ban fejlesztenünk – mindenesetre, érdemes elgondolkodni ezen a következő feladat előtt.

Linux Journal 2006., 145. szám

Gene Sally tíz éve dolgozik Linuxszal, különféle munkák keretében. Gene mostanában a beágyazott rendszerekkel foglalkozó mérnökök segítésére összpontosít. Bátran írjon neki a gene.sally@gmail.com címre, ha kérdése van.

Idő van!

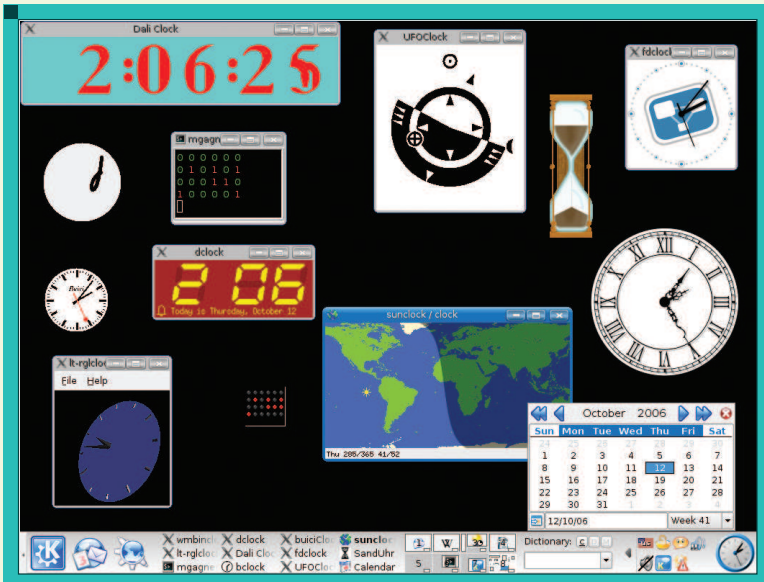
Van ugyebár az a bizonyos klasszikus kérdés, mely szerint tudja egyáltalán valaki, hogy pontosan hány óra van. Nos, úgy tűnik kedvenc szakácsuk válasza „igen”, viszont ehhez az emberek rengeteg órával kell rendelkeznie...

Az idő, ***Farncois***... Bizony mondom néked, minden az időn múlik. Igen, később kicsit részletesebben is kifejtem majd, miről is filozofálok, most viszont hajrá, mert az idő szalad, a vendégeink pedig mindjárt itt lesznek. Ellenőrizzük, hogy a fő kiszolgálónk órája szinkronba van-e referenciaidőnkkel! Na, de felejtsük el az egészet. Egyrészt teljesen biztos vagyok benne, hogy pontosan jár az az óra, másrészt meg megjöttek a vendégeink. Irány tehát a pince. Hajrá! Menj a déli szárnyba kérlek, és hozd fel nekünk azt a 2001-es ***Chateauneuf du Pape*** bort, amit ma este kóstolgattunk kicsit nyitás előtt. Én meg addig leültetem a vendégeinket.

Isten hozott tehát mindenkit ***Marcel*** vendéglőjében, ahol a jó borok a kiváló linuxos és nyílt forrású alkalmazásokkal találkoznak. Na és persze a társaság is legkiválóbb a világon. Kérem üljenek le asztalaikhoz, és helyezzék magukat kényelembe. Már leküldtem ***Francoist*** a pincébe, hogy hozza fel nekünk a mai napra kiválasztott bort. Azt hiszem már hamarosan vissza kell érnie. Ha vetnek egy pillantást az asztalfelületre, felfedezhetik, hogy mindenféle időmérő eszközök sorakoznak ott, mégpedig szép számban. Ez igényel némi magyarázatot, tehát azt hiszem az lesz a legjobb, ha egy kis mozitörténelemmel kezdem a mai bemutatót.

Bátorkodom feltételezni, hogy akad önök között olyan, aki kellően idős ahhoz, hogy emlékezzen ***H. G: Wells „Az időgép”*** (***The Time Machine***) című regényének 1960-as, ***George Pal*** által rendezett filmváltozatára. A filmben ***Georgenak*** – akit különben maga Wells játszott – van egy szobája telis-tele órákkal. Volt ott mindenféle óra. Kakukkos órák, nagyapáinktól örökölt zsebórák meg minden, épp csak digitális óra nem volt egy sem. Nos, ha valakinek túl régi ez a film ahhoz, hogy emlékezhessen rá, akkor talán fel tudja idézni ***Robert Zeneckis*** 1985-ös ***„Vissza a jövőbe”*** (***Back to the Future***) című alkotását, amelyben a főszerepet ***Michael J. Fox*** játszotta. ***Brown*** doktornak – akit ***Christopher Lloyd*** alakított – szintén volt egy laborja tele hagosan ketyegő órákkal. Namost akkor próbáljuk meg kitalálni, vajon melyik forgatókönyvíró inspirálta a másikat... Vessünk talán egy pillantást az *1. ábrára* kedveseim, és máris felfedezhetjük a ***Linux*** asztalfelületén ***George*** viktoriánus stílusú szobáját, vagy ***Brown*** doki laborját, attól függően, hogy ki mire emlékszik szívesebben.

Bizonyára sokan vannak olyanok, aki most azt kérdezik, ugyan miért akarna bárki még egy órát telepíteni a rendszerére. Elvégre a ***KDE*** és ***GNOME*** egyaránt tartalmaz egyet a panelbe építve. Minek még egy? Aztán meg kattintsunk csak arra az órára, és lám, egy szép kis naptár bukkan elő, amint az az általam készített képernyőmentésen is látható (*1. ábra*). Na, de akárhogy is van, az órák nagyszerű dolgok, sőt az egyik nagyszerűbb mint a másik. Éppen ezért ma mindenféle órákat fogok bemutatni az önök szórakoztatására. Lesz itt pár eszeveszettül „megdizájnolt” darab, sőt előfordul majd pár kifejezetten fura is. Azt gondolom, hogy a kínálat kellően széles lesz ahhoz, hogy mindenki megtalálja magának azt, ami leginkább megy az egyéniségéhez. Na, de itt van valami, ami garantáltan tetszeni fog mindenkinek. A mi kedves pincérünk ***Francois*** éppen visszaért a pincéből a borral! Nosza ***Francois***, kérlek tölts a vendégeinknek!

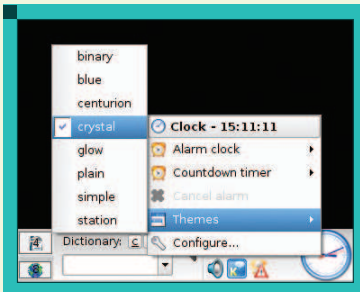


■ ***1. ábra*** Vajon Marcel tényleg pontosan tudja, hány óra van, ha ennyi eszközzel méri egyszerre?

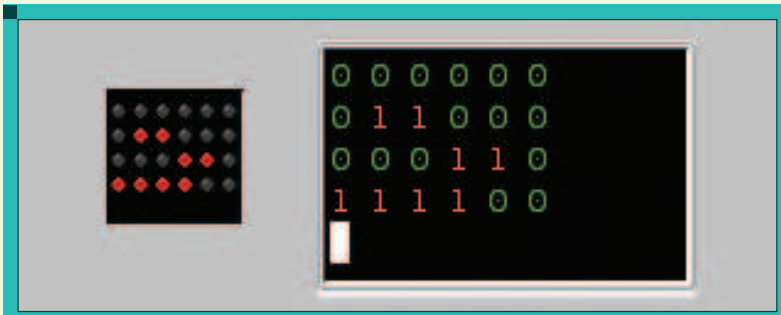
Amíg ***Francois*** kitölti a bort, addig mi vessünk még egy pillantást arra a bizonyos jobb alsó sarokban levő órára. Az ott nem a ***KDE*** alapértelmezett órája, hanem ***Fred Schattgen StyleClock*** nevű programja, ami nevének megfelelően egy mindenféle témákkal átszabható változata az eredetinek. Van enne például ébresztőóra meg visszaszámláló is. (Az önök kedvenc szakácsa is ezt használja, ha főzés közben ejtőzni szeretne egy kicsit.) A menüből beállíthatjuk az ébresztő funkciót vagy a visszaszámlálót is. Mindkét üzemmódhoz tartozik egyetlen kattintással beállítható személyes alapértelmezés, de természetesen bármilyen értéket megadhatunk, különben az egész dolognak nem lenne értelme. Az meg szinte magától értetődik, hogy különböző témákat is beállíthatunk az órához. Jómagam az analóg kijelzési módba vagyok beleszeretve, de a ***StyleClock*** rendelkezik digitális kijelzési módokkal is. Ezek között szerepel a „kockáknak" tervezett és kötelezően használandó bináris óra. Ha már a bináris óráknál tartunk, aki régóta olvassa a rovatomat, az bizonyára emlékszik rá, hogy bár leginkább ***KDE*** rendszert használok, azért a ***Window Maker*** ablakkezelő is a kedvenceim köz tartozik azokkal a bizonyos beépülő alkalmazásaival (***dock apps***). Éppen ezért a következő óratípus, amit be szeretnék mutatni a ***Thomas „Engrim" Kuiper*** és ***Sune Fjord*** által készített ***wmBinClock***. Ez az apró ***Windows Maker*** beépülő alkalmazás egyaránt képes az időt vízszintesen és függőleges megjeleníteni. Természetesen a vízszintes megjelenítés az alapértelmezett. Ha az ember kíváncsi a pontos időre, csak vet egy pillantást a biteket szimbolizáló ***LED***-ekre, átfordítja – szigorúan fejben – binárisról decimálisra a látottakat, és örül. (Digitális 1-nek a ***LED*** bekapcsolt, nullának a kikapcsolt állapota felel meg.) Ha az óra „elrendezése" vízszintes, akkor a másodperceknek az első két függőleges ***LED***-sor felel meg a jobb oldalon. A két középső blokk szimbolizálja a perceket és így tovább. Egyszóval ez az alkalmazás valami fantasztikus, és nem, nem kell ***Window Makert*** használnunk ahhoz, hogy kipróbálhassuk. Ugyanolyan jól működik ***KDE*** és ***GNOME*** alatt is. No, de biztosan akadnak olyanok is az olvasók között, akik nem grafikus felületet használnak, vagy egyszerűen csak jobban szeretnek dolgokat egy terminálablakban „elintézni". Bizonyára ők is szeretik a bináris órákat... Nekik való tehát ***Nico Golde BinClock*** nevű programja, amely egy terminálablakban jeleníti meg a pontos időt, továbbra is bináris formában. Alapértelmezésként a kijelzés módja teljesen megegyezik a ***wmBinClock***-nál bemutatottal (*3. ábra*), de természetesen itt nem folyamatos, hanem egyszeri kijelzésről van szó. Ha mégis folyamatosan akarjuk futtatni a parancssori órát, akkor használjuk a `-l` parancssori kapcsolót:

```
binclock -l
```

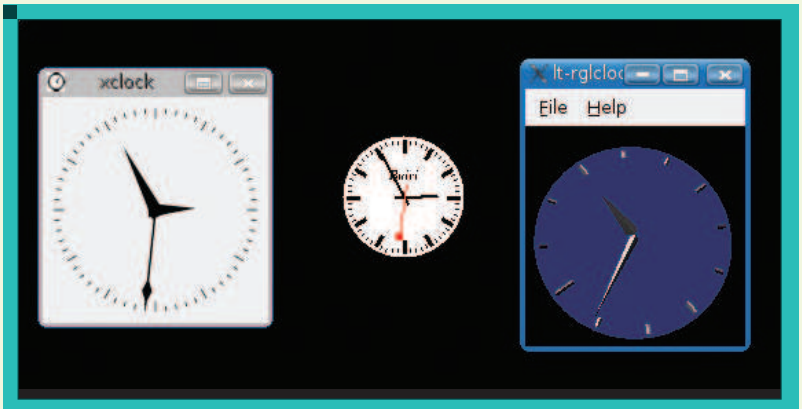
A parancssori kapcsolók listáját a `-h` kapcsolóval jeleníthetjük meg. Van opció az egysoros megjelenítéshez, a nullák és egyek színének beállításához, sőt még a hagyományos megjelenítésre való visszaálláshoz is. Persze ha valakinek egyszerűen az idő szöveges kijelzésére van szüksége, nyugodtan használhatja a szabványos `date` parancsot is. Ha pedig az adott hónap



■ ***2. ábra*** Ha át akarjuk szabni a StyleClock kinézetét, csak kattintsunk rajta a jobb egérgombbal.



■ ***3. ábra*** Két bináris kijelzést használó óra: a wmBinClock és a dclock. Úgy fest egész jó a szinkron...



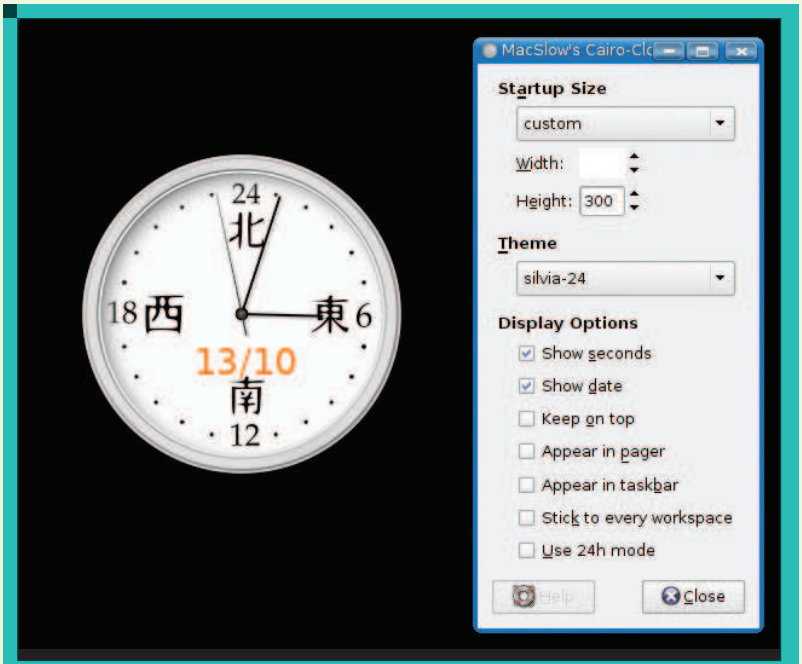
4. ábra Az analóg órák a legkülönbözőbb stílusokkal rendelkezhetnek kezdve az Xclock mára már klasszikus kinézetétől, az egyszerű de nagyszerű Buici clock-on át (középen), egészen a forgómorgó rgclock-ig (jobbra).

naptárát akarjuk megjeleníteni, adjuk ki a `cal` parancsot. Én azonban a továbbiakban is inkább az érdekességekre szeretnék koncentrálni.

Clockywock

Ha már a különlegességeknél tartunk, el nem mulasszuk kipróbálni ***Thomas S. Glascock Clockywock*** nevű programját, amit a ➲ ***www.soomka.com*** címen találunk. Ez egy az ***ncurses*** könyvtárra támaszkodó analóg óra, ami ennek megfelelően terminálablakban képes futni. Az egész olyan, mint mikor találkozik egymással a ma és a tegnap technológiája.
Bár a bináris órák sokakat elkápráztatnak, azért van valami megkapó egy retro érzést árasztó, analóg órának a felhasználói felületen való felbukkanásában is. Aki pedig meg szeretné tapasztalni ezt az érzést, annak első közelítésben nem kell semmit letöltenie vagy telepítenie, hiszen ott az ***Xclock***, ami az ***X Window*** rendszer részeként települ minden rendszerre. Ezt az apróságot eredetileg ***Tony Della Fera***, ***Dave Mankins*** és ***Ed Moy*** írták. Futtatásához nem kell egyebet tennünk, csak begépelni az `xclock` parancsot egy terminálban. (Avagy használhatjuk az ***Alt+F2*** billentyűkombinációt is, ami grafikus felületen ad lehetőséget efféle parancsok futtatására.) Alapértelmezésként a program nem jelenít meg másodpercmutatót, de azért van ilyen funkciója. A bekapcsolásához használjuk az `xclock` ***-update 1*** parancsot. Ez megjeleníti a másodpercmutatót is, ami – minő meglepetés – minden másodpercben ugrik egyet előre.

Az ***Xclock*** nem sokat változott az évek során, ami különben érthető is, elvégre minek megjavítani valamit, ami úgy jó, ahogy van. Ugyanakkor a teljes változatlan egyeseket, például Marc Singert arra indította, hogy megírja a ***Buiuci clock*** nevű alkalmazást. Igazából ebben a programban sincs semmi rendkívüli. Egy grafikailag szépen kivitelezett analóg óra szép piros másodpercmutatóval. Ennyi. Azoknak, akik ennél többre vágynak az animáció terén ***Kaz Sasayama rgclock*** programját tudom ajánlani. Ez egy forgó ***3D Mesa/OpenGL*** alapú óra, amit ez egérrel megfogva elhúzhatunk bárhova a képernyőn, és a forgási sebességét és irányát is szabadon befolyásolhatjuk. Ezt a három óraprogramot mutatja a *4. ábra*.
Nem kétséges, hogy ***Mirco Muellernek*** a még kifinomultabb megjelenés elérése volt a célja akkor, amikor megírta a ***Cairo Clock*** nevű programot. Komolyan mondom, ez az alkalmazás a maga nemében bámulatos, már ami a kinézetét illeti. Van neki számos különböző megjelenítési módja, például 12- és 24 órás is, hogy csak valami egészen alapvetőt említsek. A futó program átállításához kattintsunk rajta a jobb egérgombbal, majd válogassunk a felbukkanó helyi menüből. Itt amúgy nem csak az óra kinézetét módosíthatjuk, hanem néhány más jellemzőjét is beállíthatjuk (*5. ábra*). Akár a méretét is átszabhatjuk, amekkorára csak akarjuk.
Nos, bár eddig az idő nagyobbik részét az analóg órák elemzésével töltöttem, azért szép számmal akadnak jó vágású digitális időmérők is. Az egyik kedvence, a ***Jamie Zawinski*** által írt ***XdaliClock*** (*6. ábra*), ami nem más, mint egy kissé furcsa kinézetű digitális óra. Furcsaságát az adja, hogy a számok rajta nem annyira átváltanak,



5. ábra A számos beállítással rendelkező, megjelenését illetően pedig kifejezetten gyönyörű Cairo Clock tulajdonságait futás közben is állíthatjuk.

5. ábra Az XdaliClock nem csak kijelzi, hanem egyenesen elolvasztja a múló másodperceket, míg a dclock kicsit szilárdabb jellem e téren.

7. ábra Az UFOClock... Első látásra nehéz eldönteni, hogy közönséges óra, vagy egy idegen faj által ittfelejtett eszköz.

inkább átfolynak egymásba. Másodpercről másodpercre, percről percre a számok elolvadnak, majd ismét összeállnak de már új formájukban bukkannak fel. Az ember úgy van vele, hogy képes kiválni az 59 perc 59 másodperc állapotot, csak hogy láthassa, amint egyszerre „folyik át" az egész számlap egy új órába. Frenetikus. Ha az `xdaliclock -cycle` kapcsolóval futtatjuk a programot, akkor az effektusban nem csak a számok, hanem a háttérszín is részt vesz.
A ***Tim Edwards*** által írt ***dclock*** – ami egyébként ***Dan Heller*** eredeti kódjának egy átirata – szintén egy kiváló digitális óra, amely a régi hétszegmenses ***LED***-es kijelzőket utánozza.
A ***dclock***-nak számos parancssori kapcsolója van, amelyekkel beállíthatjuk például a dátum formátumát, a ***LED***-ek színét (a bekapcsolt és kikapcsolt állapotot egyaránt), és számos egyéb dolgot. A következő paranccsal például

```
dclock -date Today is %A, %B %d
➥-fg yellow -bg brown -led_off
➥brown4
```

a *6 ábra* alsó részén látható órát tudjuk „előállítani". A számos parancssori kapcsoló mellett a programnak interaktív szolgáltatásai is vannak. Amíg fut, addig elegendő az egérmutatót az ablakába húzni ahhoz, hogy egybetűs billentyűparancsokkal irányíthassuk. Az S lenyomásával például bekapcsolhatjuk a másodpercek kijelzését, az R megcseréli az előtér és háttér színét, a / pedig növeli a számjegyek dőlésszögét. Ezekről a lehetőségekről természetesen teljes leírást találunk a program dokumentációjában.
Nos, úgy tűnik annyit beszéltünk ma már az órákról, hogy egészen elszaladt az idő és közeleg a záróra. Míg ***Francois*** még egyszer újratölti mindenki poharát, én mesélek még egy kicsit, mégpedig talán minden idők legfurcsább órájáról, a ***Matt Wronkiewi*** által készített ***UFOClockról*** (*7. ábra*). Kinézete alapján azt hiszem ez a program méltó vetélytársa lehet az előzőekben bemutatott időmérő eszközöknek, de legalábbis megérdemel némi helyet az asztalfelületen.
Az ***UFOClock*** megjeleníti a szokásos időt, de emellett kijelzi, hogy épp milyen fázisánál tart a ***Hold***, aktuálisan mennyi a nappal és az éjszaka hosszának aránya, mennyi az alkonyatig vagy a napkeltéig hátralevő idő, és hogy mennyit kell még várnunk a következő napfordulóig illetve napéjegyenlőségig. Namost ha valaki azt akarja kérdezni, hogy én ezt mind értem-e, akkor a válaszom az hogy majdnem. Azt hiszem még rágódok a dolgon egy kicsit. A telepíthető csomagban találunk egy mintát a konfigurációs fájlra is. Ez azért fontos, mert ebben kell megadnunk, hogy milyen hosszúsági és szélességi körön tartózkodunk. (Ez ugyebár nem nélkülözhető ahhoz, hogy megmondjuk, hány óra van.)
Most pedig kedveseim hogy úgy mondjam tényleg idő van. Akárhány órán nézzük is egyszerre az idő múlását, nem szakadhatunk el a valóságtól: sajnálom, de záróra van hölgyei és uraim, pedig még annyi szép órát fedezhettünk volna fel együtt.
De sebaj, emeljük poharunkat és igyunk egymás egészségére!
A ***votre sante, Bon appetit!***

Tipp
Aki még több órát szeretne látni, az látogasson el ***Marcel Mester*** webhelyére, egész pontosan a ➲ ***www.marcelgagne.com/clocks.html*** címre, ahol további fantasztikus darabokat találhat.

Linux Journal 2007., 153. szám

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

KAPCSOLÓDÓ CÍMEK

➲ www.linuxjournal.com/article/9456

Linux és Windows egy gépen

Hogyan használjunk mindkét platformon elérhető alkalmazásokat úgy, hogy a szükséges adatokat csak egyszer tároljuk egy a Linux és a Windows számára egyaránt elérhető helyen...

Nemrég úgy döntöttem, hogy a *Toshiba* notebookomat átalakítom úgy, hogy *Windows* és *Linux* egyaránt legyen rajta. A két telepített operációs rendszer a *Windows XP Professional* és az *Ubuntu Linux* volt. Mindezt abban a reményben tettem, hogy a mindkét rendszer alatt elérhető alkalmazásokat – ilyen például a *Mozilla Firefox*, a *Mozilla Thunderbird*, az *AbiWord*, a *Gnumeric* vagy a *SciTE* – képes leszek majd transzparens módon használni, vagyis úgy, hogy mindig ugyanazokat a beállításokat láthassam, függetlenül attól, hogy bootolásnál melyik menüpontot választottam. Ebben a cikkben azt írom le, hogyan sikerült ezt végül megoldanom.

Két operációs rendszer közös alkalmazásokkal

A továbbiakban eleve feltételezem, hogy az olvasó rendelkezik egy olyan géppel, amelyen kifogástalanul működik egy *Windows* és egy *Linux* operációs rendszer. Szintén szükségünk lesz egy megfelelő méretű kiegészítő partícióra, amelyen a megosztott alkalmazásokat fogjuk tárolni. Ezen a lemezterületen értelemszerűen olyan fájlrendszernek kell lennie, amit mindkét operációs rendszer gond nélkül tud írni és olvasni is. A legkézenfekvőbb választás természetesen a *FAT32* (*VFAT*). Az én laptopomban egy 30 GB-os merevlemez van, amire a forgalmazó eleve föltelepítette a *Windows XP Professional* változatát. Amikor ebbe a bizonyos manőverbe belekezdtem már viszonylag régóta használtam a gépet, és a lemez is csaknem tele volt hasznos adatokkal. Első lépésként tehát lementettem amit tudtam, és a *Windows* töredezettség-mentesítőjével annyira rendbe raktam a maradékot, hogy a *Windows* által elfoglalt terület 15 GB alá menjen. Ezután a *Linux System Rescue CD*-n található segédeszközökkel átméreteztem a *Windows* partícióját, majd létrehoztam azokat az új lemezterületeket, amelyekre a *Linux* telepítéséhez szükség volt. Egész pontosan a következő rendszert alakítottam ki:

1. partíció: *Widows NTFS* elsődleges fájlrendszer, 18,5 GB

2. partíció: *Linux ext3* fájlrendszer, 5 GB

3. partíció: *Linux* csereterület (*swap*), 1 GB

4. partíció: *FAT32* fájlrendszer a közös alkalmazások számára, 5 GB

Az az igazság, hogy egy mindössze 30 GB-os merevlemezen két operációs rendszert futtatni nem éppen ideális. Miután visszatöltöttem az archivált leveleimet a megosztott partíció úgy 80 %-ig megtelt. Éppen ezért azt javaslom mindenkinek, aki hasonló rendszert készül kialakítani, hogy legalább egy 60-80 GB-os merevlemezzel próbálkozzon. Ha nekem lett volna ilyen lemezem, akkor valószínűleg hagyok 20 GB-ot a *Windows*-nak, 10 GB-ot a *Linuxnak*, 1-2 GB-ot a csereterületnek, a maradékot pedig közös területként használnám FAT32 fájlrendszerrel.

A közös partíció beállítása és kezelése

A *Windows* a *FAT32* partíciókat – köztudomásúlag – külön meghajtóként látja és egy betűjelet rendel hozzájuk. Hogy ez a betűjel pontosan mi is lesz, az attól függ, hogy milyen egyéb eszközök – floppy, *CD*/*DVD* olvasó stb. – csatlakoznak a rendszerhez. Az én esetemben a „közterület” az E: jelet kapta *Windows* alatt. Ha valaki esetleg nem tudná az efféle betűjeleket a *Windows Explorer* segítségével lehet felderíteni.

Amikor telepítettem az *Ubuntu Linuxot* azt úgy állítottam be, hogy a *FAT32* partíciót már bootolás közben csatolja be a rendszerbe, mégpedig a */share* könyvtár alá. Hogy ez valóban megtörtént-e, azt a bootolás után például a *UNIX* `df` parancsával ellenőrizhetjük (lásd az *1. Listát*).

Bár a */share* partíciót a rendszer sikeresen becsatolta, azért van egy kis probléma. Alapértelmezésként ennek a fájlrendszernek a tulajdonos a root felhasználó, ami azt jelenti, hogy egy közönséges felhasználó semmit se tud vele kezdeni, hiszen irási és olvasási joga a rajta tárolt adatokhoz. Ez egyben azt is jelenti, hogy programokat sem fog tudni innen futtatni. Szerencsére a *UNIX* `mount` parancsának mindenféle paramétert át lehet adni, így például azt is beállíthatjuk, hogy egy fájlrendszernek ne a root, hanem valamelyik közönséges felhasználó legyen a tulajdonosa. Ez az egyik legegyszerűbb módja annak, hogy egy egyszerű felhasználói fiók tulajdonosának hozzáférést engedjünk a megosztott partícióhoz.

1. Lista – A UNIX df parancsa szerint a /share partíció él és mozog

```
kevin@lyratoshibaubuntu:~$ df -k
Filesystem   1K-blocks      Used Available Use% Mounted on
/dev/hda2      5036316   1748816   3031668 37%  tmpfs
/dev/shm        184936         0    184936  0%  tmpfs
/dev/hda1       184936     12588    172348  7% /lib/modules/
                                              2.6.12-9-386/
                                              volatile
/dev/hda1     18427896   9955608   8472288 55% /media/hda1
/dev/hda4      4713876    417898   4295978  9% /share
```

Ha tehát csak egy felhasználó fogja a gépet használni, vagy csak egy felhasználónak van szüksége a megosztott alkalmazásokra, akkor a legcélravezetőbb megoldás az, ha a `mount` megfelelő paraméterezésével egyszerűen őt tesszük meg a */share* partíció tulajdonosának. Ehhez tudnunk kell az illető felhasználói és csoportazonosítóját. Ezt az információt a */etc/passwd* fájlban találjuk meg. Íme ennek a fájlnak egy részlete az én gépemről (a bejelentkezési nevem kevin):

```
kevin@lyratoshibaubuntu:~$
cat /etc/passwd | grep kevin
kevin:x:1000:1000:kevin,,,:/
➥home/kevin:/bin/bash
```

A numerikus felhasználói azonosító a második kettőspont után látható szám. Hasonlóan a csoportazonosító a harmadik kettőspont után szerepel. A fenti példa szerint a kevin nevű felhasználó azonosítója és csoportazonosítója egyaránt 1000.

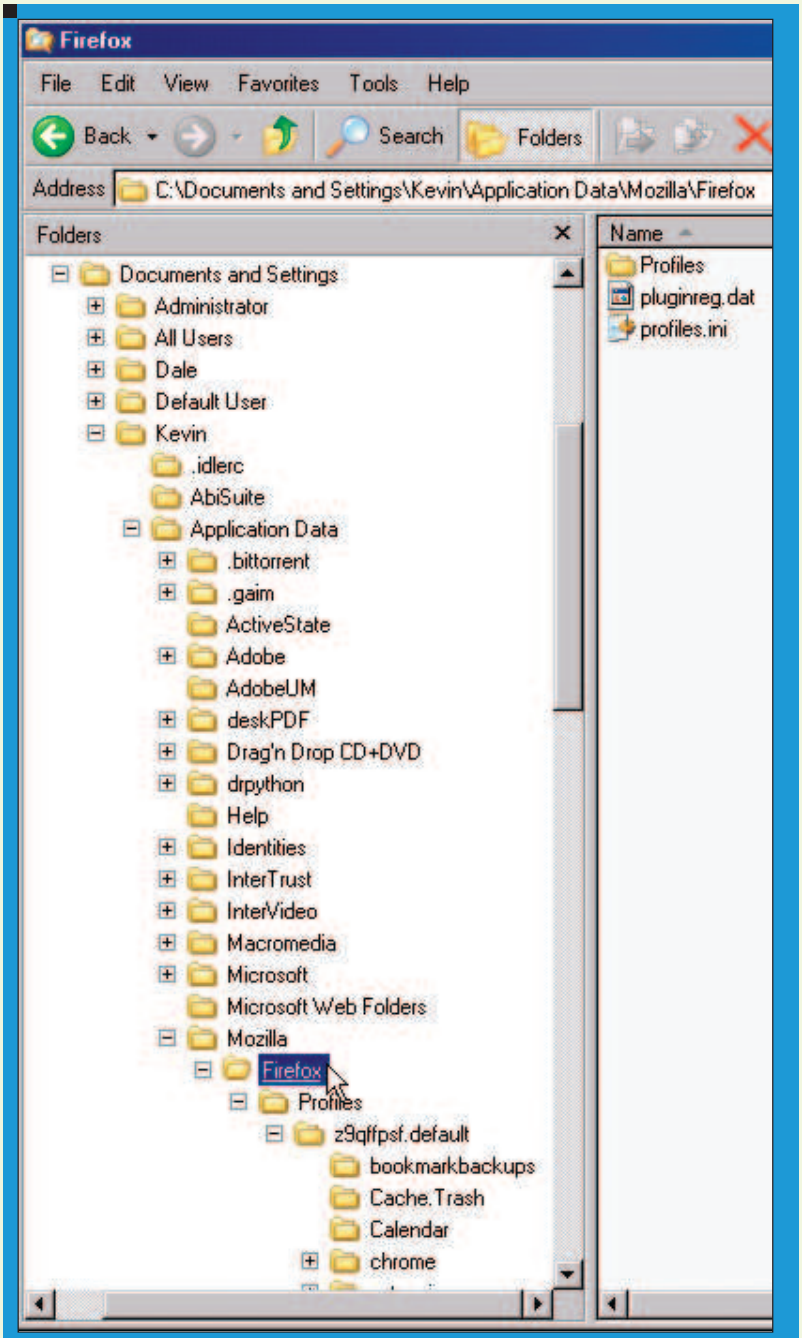
A következő lépésben megfelelően át kell szerkesztenünk a */etc/fstab* fájl tartalmát. Ebben a fájlban azok a fájlrendszerek vannak felsorolva, amelyeket a *Linux* rendszernek bootolás közben – jó esetben – látnia kell. Szintén meg vannak adva benne azok a műveletek, amelyeket ezekkel a fájlrendszerekkel el kell végeznie. Ahhoz, hogy ezt a fájlt szerkeszteni tudjuk, át kell váltanunk rendszergazdai módba. Mielőtt bármit csinálnánk, először is készítsünk egy másolatot a */etc/fstab* fájlról. Ez még jól jöhet, ha valamit elrontottunk, és vissza szeretnénk állítani az eredeti, működő állapotot. Ha ez megvan, akkor nyissuk meg a fájlt egy tetszőleges szövegszerkesztővel – ilyen például a *vi*, az *emacs*, a *gedit* vagy a *scite* – keressük meg a megosztott fájlrendszernek megfelelő sort, és írjuk át az `<options>` oszlopban található opciókat megfelelő módon. Ez bővebben azt jelenti, hogy a `defaults` kulcsszó után írjuk be az `uid=uuuu` és a `gid=gggg` opciókat ahol `uuuu` és `gggg` a */etc/passwd* fájlból kinézett felhasználói és csoportazonosítót jelöli. Ha megvagyunk, akkor a */etc/fstab* fájlnak valahogy úgy kell kinéznie, ahogy azt a 2. *Listában* láthatjuk.
Ha több felhasználói fióknak is hozzáférést kell adnunk a megosztott partícióhoz, akkor más stratégiát kell választanunk. Az egyik megoldás, hogy a */share* partíciót nem bootolás közben csatoljuk be, hanem írunk egy olyan szkriptet, ami elvégzi ezt. Ezt lefuttatva a felhasználók kiszolgálhatják magukat, vagyis beemelhetik a kérdéses lemezterületet úgy, hogy tulajdonosi joguk legyen fölötte, és teljes hozzáférésük a tartalmához.
Ahhoz, hogy megakadályozzuk a rendszerindítás közben történő becsatolást, nyissuk meg a */etc/fstab* fájlt és tegyük egy # karaktert a kérdéses sor elé. Ezzel az adott sor megjegyzésé alakul, vagyis a rendszer figyelmen kívül fogja hagyni. Ezután keressük ki a */etc/passwd* fájlból mindazoknak a felhasználóknak a felhasználói és csoportazonosítóját, akiknek hozzá kell férniük a megosztott fájlrendszerhez. Aztán írjunk meg egy a következőhöz hasonló szkriptet, és helyezzük el a másolatait minden érintet felhasználó saját könyvtárában. Ügyeljünk rá, hogy minden példányba a megfelelő azonosítók kerüljenek az `uid=` és a `gid=` rész után:

```
kevin@lyratoshibaubuntu:~$ cat
➥mountShare.csh
sudo mount -t vfat -o uid=1000,
➥gid=1000 /dev/hda4 /share
```

Miután bejelentkezett a rendszerbe, a felhasználónak nyitnia kell egy terminálablakot, és végre kell hajtania

2. Lista – A /etc/fstab fájl tartalma a módosítás után. A megosztott partíció opcióhoz hozzáfűzött részek gondoskodnak róla, hogy becsatolása után tulajdonosa a megadott felhasználó legyen.

```
kevin@lyratoshibaubuntu:~$ cat /etc/fstab
# /etc/fstab: static file system information.
#
# <file system> <mount point>   <type>      <options>                <dump>  <pass>
proc            /proc           proc        defaults                  0       0
/dev/hda2       /               ext3        defaults,errors=remount-ro 0      1
/dev/hda1       /media/hda1     ntfs        defaults                  0       0
/dev/hda4       /share          vfat        defaults,uid=1000,gid=1000 0      0
/dev/hda3       none            swap        sw                        0       0
/dev/hdc        /media/cdrom0   udf,iso9660 user,noauto               0       0
```



1. ábra A Firefox beállításait tartalmazó könyvtárszerkezet Windows alatt

a fenti szkriptet a megosztott *FAT32* fájlrendszer megfelelő jogosultságokkal való beemeléséhez.

```
bash ./mountShare.csh
```

Ha sikeresen becsatoltuk a fájlrendszert, egy `ls -l` paranccsal ellenőrizhetjük, hogy valóban megvannak-e vele kapcsolatban a megfelelő jogosultságaink:

```
kevin@lyratoshibaubuntu:~$ ls
➥-l / | grep share
drwxr-xr-x   18 kevin kevin
➥4096 1969-12-31 19:00 share
```

A megosztott terület használata

Ezzel gyakorlatilag készen is állunk arra, hogy bortokba vegyük a rendszert, és feltöltsük a közös területet olyan alkalmazásokkal, amelyek mind *Linux* mind *Windows* alatt működnek. Ide kerülhetnek természetesen azok az adatok is, amelyeken ezekkel az alkalmazásokkal dolgozni szeretnénk. A továbbiakban csak arra kell ügyelnem, hogy ha *Windows* alatt dolgozom, akkor valamennyi dokumentumokat és adatomat a E: meghajtóra mentsem. Itt talán nem árt megismételni, hogy az Olvasó gépén ez a betű más is lehet attól függően, hogy hány meghajtót kezel az operációs rendszer. Hasonlóan ha *Linux* alatt dolgozom, akkor a */share* partíción levő könyvtárakban kell elhelyeznek az anyagaimat ahhoz, hogy később *Windows* alól is láthassam őket.
Van még egy dolog, amire ügyelnünk kell, mielőtt elkezdjük élesben használni a rendszert. Győződjünk meg róla, hogy *Linux* és *Windows* alatt az alkalmazásoknak ugyanaz a változata fut. Abban pedig nem reménykedjen senki, hogy az egyes változatok között nincsenek olyan eltérések, amelyek a beállítófájlok vagy az alkalmazásokkal létrehozott dokumentumok belső szerkezetét érintik. Lesznek.

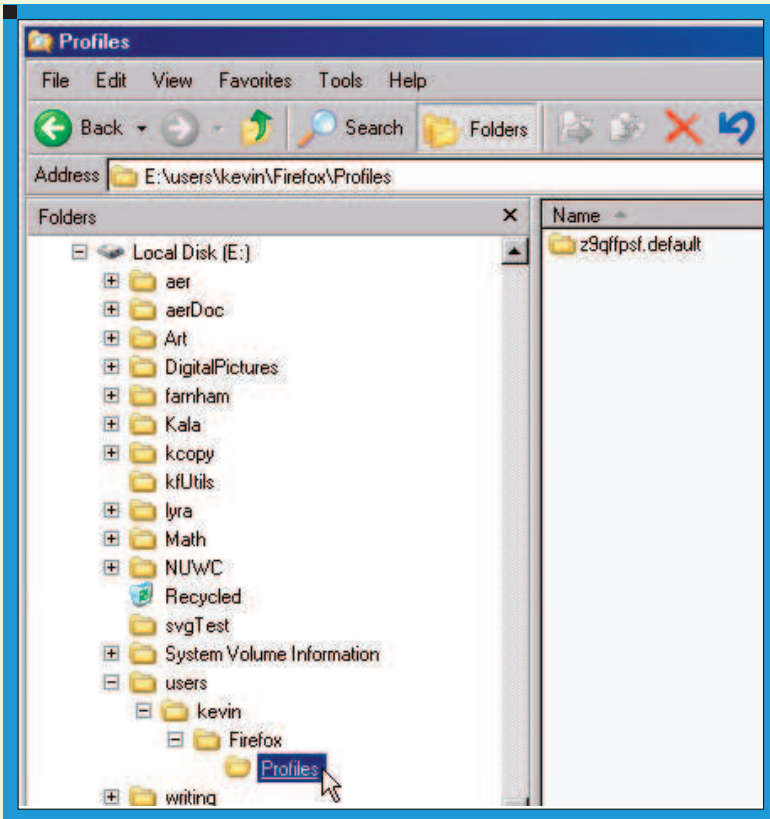
A Mozilla csomag

Jómagam webböngészőként a *Mozilla Firefoxot*, levelezőkliensként pedig a *Mozilla Thunderbird* programot használom. Mikor föltelepítettem a laptopomra a *Linuxot*, már régóta használtam a *Firefoxot*, így rengeteg értékes könyvjelzőm volt. A levelezéssel hasonló volt a helyzet. Évek „termése" volt már a különböző mappákban. Az új rendszert természetesen úgy akartam kialakítani, hogy mindkét operációs rendszer alól zökkenőmentesen tudjam használni mindezeket. *Linux* és *Windows* alatt is látni szerettem volna a *Firefoxból* azokat a könyvjelzőket, és mindkét rendszer alatt olvasni akartam a *Thunderbirddel* az archivált leveleimet is.
Lehetséges ez egyáltalán? A *Mozilla Suite* fejlesztői által követett konfigurációs stratégiának köszönhetően a válasz egyértelműen igen! Mind a *Firefox*, mind a *Thunderbird* profilokon keresztül oldja meg a beállítási adatok tárolását. Minden profil külön alkönyvtárban kap helyet, amelyek alapértelmezésként közvetlenül az alkalmazás legfelsőbb szintű beállítási könyvtára alatt kapnak helyet. Ezen kívül valamennyi profil neve és helye össze van gyűjtve egy indexfájlban,

amit *profiles.ini*-nek hívnak. A beállítási információk tárolásának ez a módja azért különösen hasznos számunkra, mert lehetővé teszi, hogy a beállításokat a fájlrendszer egy tetszőleges olyan pontján tároljuk, amelyhez az alkalmazást futtató felhasználónak írási és olvasási joga van. Ez pedig lehet például az a bizonyos sokat emlegetett megosztott *FAT32* fájlrendszer.
Mielőtt bármit megváltoztatnánk a beállításokon, győződjünk meg róla, hogy sem a *Firefox*, sem a *Thunderbird* nem fut. Aztán hozzunk létre egy olyan könyvtárat a megosztott fájlrendszeren, amelyben a *Mozilla* csomag beállításait fogjuk tárolni mind a *Windows*, mind a *Linux* alatt futó változat számára. Én úgy döntöttem, hogy létrehozok egy *users* nevű könyvtárat, ez alatt pedig egy kevin nevű alkönyvtárat, tekintettel hogy mindkét rendszer alatt ez a bejelentkezési azonosítóm. Ez a módszer akkor is megfelelő, ha csak később derül ki, hogy mégis több felhasználó fogja használni a gépet. Ebben az esetben sincs ugyanis más dolgunk, mint a *users* könyvtár alatt létrehozni minden egyes újabb felhasználói fiókhoz egy újabb alkönyvtárat, ahol a kérdéses felhasználó a saját beállításait tárolhatja.
Windows alatt tehát a beállításaimat tároló könyvtár elérési útvonal *E:\users\kevin*. *Linux* alatt ugyanezt a könyvtárat a */share/users/kevin* helyen találom meg.

A Mozilla Firefox megosztott beállításai

Talán érdemes megemlíteni, hogy én az itt bemutatott beállításokat a *Firefox 1.5*-ös változatával végeztem el, maga az eljárás azonban teljesen hasonló az 1.0.x-es sorozatnál is.
Mivel a *Firefoxot* eddig *Windows* alatt használtam, értelemszerűen ott volt valamennyi személyes könyvjelzőm és minden más beállításom. Azzal kezdtem tehát, hogy a *Windows* beállításait módosítottam a közös működés feltételeinek megfelelően. Ehhez először is a *Documents and Settings* mappában keressük meg a felhasználói nevünket, majd abba az *Application Data* nevű alkönyvtárat. Itt lennie kell egy *Firefox* nevű könyvtárnak, abban pedig egy *Profiles* nevű alkönyvtárnak. A *Profiles*-nak szintén kell legyen legalább egy alkönyvtára.

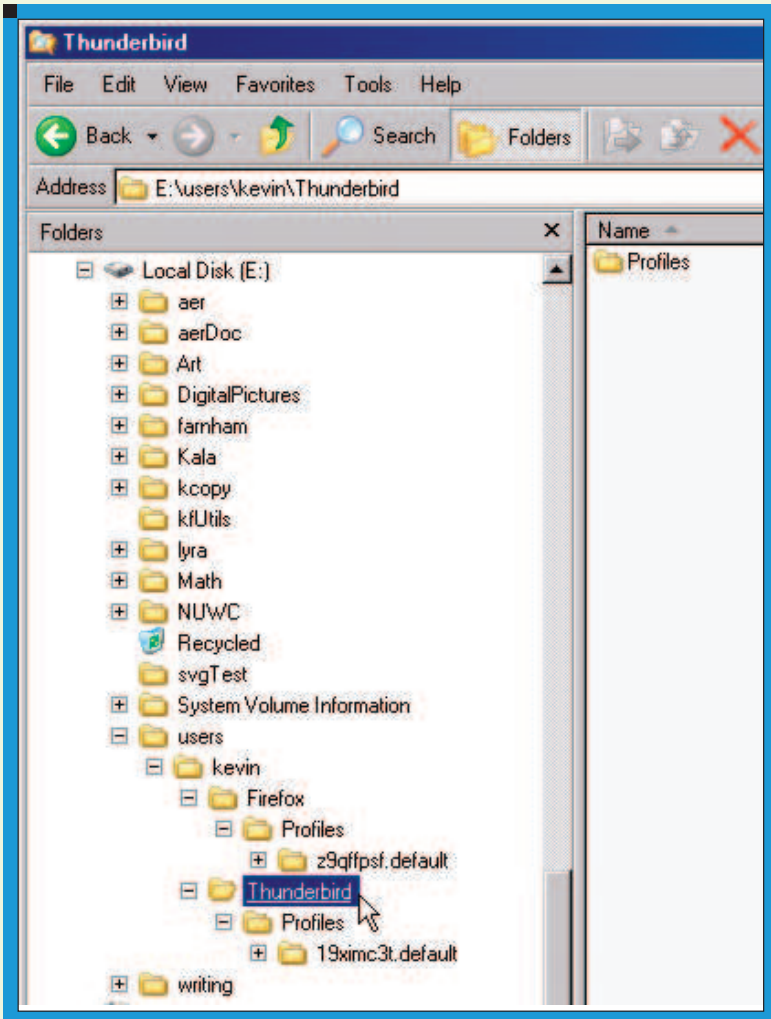


2. ábra A Firefox konfigurációs könyvtárszerkezete immár az új helyen

Az én rendszeremen mindez az *1. ábrán* látható módon nézett ki. A *profiles.ini* nevű fájl alapján tudja a *Firefox*, hogy hol keresse a beállításokkal kapcsolatos adatokat. Nyissuk meg tehát a *profiles.ini*-t egy szövegszerkesztőben, mert át kell benne írnunk néhány dolgot. Körülbelül a következő tárul majd a szemük elé:

```
[General]
StartWithLastProfile=0
[Profile0]
Name=default
IsRelative=1
Path=Profiles/z9qffpsf.default
Default=1
```

A fenti *profiles.ini* szerint az én beállításaim nem valami összetettek: mindössze egyetlen profilom van, amelynek a leírását a *Profiles/z9qffpsf.default* könyvtárban kell keresni, mégpedig ahhoz a könyvtárhoz viszonyítva amelyben maga a *profiles.ini* fájl található. Ha vetünk egy pillantást az *1. ábrán* látható könyvtárszerkezetre, akkor láthatjuk is ezt a bizonyos *Profiles/z9qffpsf.default* nevű könyvtárat, amiből számos alkönyvtár nyílik. Nos, ezek azok, amelyekben az én egyedi *Firefox* beállításaim tárolódnak. Ezek azok az adatok, amelyekhez *Windows* és *Linux* alól is szeretnék hozzáférni, mégpedig úgy, hogy írni és olvasni egyaránt tudjam őket mindkét helyről. Szeretném megint hangsúlyozni, hogy az Olvasó gépén a **.default* könyvtár nevének eleje más lesz, vagyis a mos következő lépésekben értelemszerűen cseréljük le a nevet a saját rendszerünkön érvényesre.
Ahhoz, hogy a konfigurációs adataimhoz mindkét operációs rendszer hozzáférhessen, létrehoztam egy *Firefox\Profiles* nevű könyvtárat a megosztott *E:\users\kevin* könyvtár alatt, majd átmásoltam bele a Firefox teljes *Profiles/z9qffpsf.default* konfigurációs könyvtárszerkezetét.
Az eredményt a *2. ábra* mutatja.
A *C:* meghajtón található *Profiles/z9qffpsf.default* könyvtár nevét ezzel egyidejűleg megváltoztattam, hogy legyen egy másolatom az adatokról arra az esetre, ha valami balul ütne ki.



3. ábra A Thunderbird áthelyezett beállítási állományai Windows alatt

A következő lépésben átírtam a ***profiles.ini*** fájlt úgy, hogy az immár az új helyre mutasson, és a Firefox a továbbiakban az „átköltöztetett" beállításokkal működjön. Ehhez az `IsRelative` jelző értékét nullára állítottam, a `Path` változóban pedig megadtam azt a teljes elérési útvonalat, ahova a beállításokat áthelyeztem. Fontos, hogy az útvonal megadásánál a ***Windows***-ra jellemző stílust használjuk, vagyis perjelek helyett backslash-t. Ellenkező esetben ugyanis a ***Firefox Windows*** alatt futó változata megmakacsolja magát, és nem hajlandó felismerni, hogy miről van szó. Ugyanilyen problémája a linuxos változatnak nincs. Összességében tehát a ***profiles.ini*** fájl az átszerkesztés után így nézett ki:

```
[General]
StartWithLastProfile=0

[Profile1]
Name=default
IsRelative=0
Path=E:\users\kevin\Firefox\
➥Profiles\ z9qffpsf.default
```

Erről a fájlról is készítettem egy másolatot ***profiles_new.ini*** néven, hogy ha valami váratlanul nem működne, akkor legyen hova visszanyúlni. Ha mindezzel elkészültünk, indítsuk el a ***Firefoxot***. Ha egy olyan ablak ugrik elő, amelyben a program azt tudakolja, hogy szeretnénk-e importálni egy másik böngészőprogram beállításait, akkor valamit bizonyosan rosszul csináltunk. Ilyenkor a ***Firefox*** felülír a ***profiles.ini*** fájlt, és létrehoz egy új könyvtárat is az alapértelmezett beállításoknak. Vegyük elő tehát a ***profiles.ini***-ről készült másolatot, vizsgáljuk meg alaposan, hogy hol lehet benne a hiba – máshol ugyebár nem lehet – és a felülírás után próbálkozzunk újra a ***Firefox*** indításával. Ha sikerült mindent jól beállítanunk, akkor a ***Firefox*** egyszerűen elindul, és pontosan ugyanúgy működik, mint ahogy eddig is tette. Látni fogjuk a könyvjelzőket, és minden egyebet, amit beállítottunk.

Nos, akkor most jöhet a linuxos oldal. A következő lépésben a ***Firefox*** nyílt rendszeren futó testvérét fogjuk beállítani, pontosabban rávenni a közös beállítások használatára. Indítsuk el tehát a Linux operációs rendszert és szükség esetén csatoljuk be a megosztott fájlrendszert a megfelelő módon. ***Linux*** alatt a ***Firefox*** egy a felhasználó saját könyvtárából nyíló ***.mozilla*** nevű rejtett könyvtárban tárolja a személyes beállításokat. Lépjünk be tehát ebbe a könyvtárba, majd ennek is a ***firefox*** nevű alkönyvtárába, és adjuk ki az `ls -l` parancsot. Itt is látni foguk egy ***profile.ini*** nevű fájlt, amely mellett most lesz egy ***pluginreg.dat*** nevű is, valamint egy a konfigurációs profilt tároló automatikusan generált nevű alkönyvtár, akárcsak Windows alatt.

Az eljárás meglehetősen hasonló az iméntihez. Ahhoz, hogy a ***Firefox*** linuxos változatát is rávegyük a megosztott partíción levő közös beállítások használatára át kell szerkeszteni a ***profiles.ini*** tartalmát. Az `IsRelative` jelzőt állítsuk megint nullára, a `Path` változó értékében pedig adjuk meg annak a ***/share*** könyvtárból nyíló helynek az elérési útvonalát, ahol a beállítások vannak. Az én esetben az előbbiekben elmondottak alapján a ***profiles.ini*** fájl tartalma az átszerkesztés után a következő volt:

```
[General]
StartWithLastProfile=1
[Profile0]
Name=default
IsRelative=0
Path=/share/users/kevin/Firefox/
➥Profiles/ z9qffpsf.default
Default=1
```

Indítsuk el a ***Firefoxot***. Ha mindent jól csináltunk, akkor egy közönséges ***Firefox*** munkamenetben találjuk magunkat, és látjuk ugyanazokat a könyvjelzőket, amelyeket ***Windows*** alatt is. Ha mégsem így történne,

akkor megint biztosak lehetünk benne, hogy a ***profiles.ini*** a ludas, ebben kell keresni a hibát. Előfordulhat persze, hogy a ***/share*** partíció jogosultságaival van a gond, vagy egyszerűen csak elfelejtettük becsatolni, de ez is könnyen ellenőrizhető. Ezen kívül már csak az útvonalat gépelhettük el. Derítsük ki tehát, hogy mi a gond, írjuk át megfelelően a ***profiles.ini*** tartalmát, aztán indítsuk újra a Firefoxot.

A Mozilla Thunderbird beállításainak megosztása

A ***Thunderbird*** beállítási adatinak szervezése meglehetősen hasonlít a ***Firefoxnál*** leírt szerkezetre. Én a program 1.0.7-es változatát használtam, de megint valószínű, hogy a többi változatnál is hasonló lesz a helyzet. ***Windows*** alatt a saját felhasználói nevükhöz tartozó ***Documents and Settings*** könyvtárban keressük meg az ***Application Data*** nevű alkönyvtárat, abban pedig a ***Thunderbird*** nevű mappát. Az ember elsőre azt gondolná, hogy a ***Firefoxhoz*** hasonlóan a ***Thunderbird*** könyvtárat is a ***Mozilla*** mappában kell keresnie, de az én rendszeremen nem ez volt a helyzet. A ***Thunderbird*** könyvtárban megtaláljuk a más ismerős ***profiles.ini*** fájlt, valamint egy ***Profiles*** nevű alkönyvtárat, éppen úgy, ahogy a ***Firefox*** esetében. Ahhoz, hogy valamennyi archivált levelüket elérhetővé tegyük ***Linux*** és ***Windows*** alól egyaránt a konfigurációt tartalmazó alkönyvtárat át kell helyeznünk a megosztott fájlrendszerre. Én ehhez létrehoztam ezen a helyen egy ***\users\kevin\Thunderbird*** nevű alkönyvtárat és átmásoltam ide a ***Profiles*** könyvtár tartalmát a ***Windows*** alatt alapértelmezett helyről. Az én áthelyezett könyvtárszerkezetemet ***Windows Explorerben*** megjelenítve a *3. ábra* mutatja.
Az eredeti beállításokat tartalmazó könyvtárat megint átneveztem hogy legyen róla másolatom, illetve hogy a ***Windows*** alatt futó ***Thunderbird*** a továbbiakban még csak véletlenül se tudjon hozzáférni.
A következő lépésben átírtam úgy a ***Thunderbirdhöz*** tartozó ***profiles.ini*** fájlt, hogy az a beállító állományok új helyére mutasson. Az eredeti ***profiles.ini*** ez én esetemben a következőképpen nézett ki:

```
[General]
StartWithLastProfile=1
[Profile0]
Name=default
IsRelative=1
Path=Profiles/19ximc3t.default
```

Ebben az `IsRelative` értékét nullára állítottam, a `Path`-ban pedig a már ismert módon megadtam a beállítások megosztott partíción levő teljes elérési útvonalát. Megint ügyelni kell rá persze, hogy a ***Windowsra*** jellemző backslash karaktereket használjuk az útvonal tagolására, de amúgy minden úgy történik, mint a Firefox esetében. A módosított ***profiles.ini*** a következőképpen nézett ki:

```
[General]
StartWithLastProfile=1
[Profile0]
Name=default
IsRelative=0
Path=e:\users\kevin\Thunderbird\
➥Profiles\ 19ximc3t.default
```

Ha ezzel is elkészültünk, akkor nincs más hátra, mint újraindítani a ***Thunderbirdöt***. Ha mindent jól csináltunk, akkor a program a megszokott módon elindul, mintha mi sem történt volna. Ha a beállítások valahol hibásak, akkor a ***Thunderbird*** nekünk szegezi a kérdést, mely szerint akarunk-e új profilt létrehozni. Természetesen nem akarunk, tehát ha ezt a kérdést látjuk, akkor nyomjuk meg a Mégsem gombot és lépjünk ki a programból. A hiba ilyenkor megint csak nagy valószínűséggel a ***profiles.ini*** fájlban keresendő, tehát nyissuk meg újra, végezzük el a szükséges módosításokat, és próbáljuk újraindítani a ***Thunderbirdöt***.
Linux alatt a ***Thunderbirdhöz*** tartozó ***profiles.ini*** fájlt a saját könyvtárunkból nyíló ***.mozilla-thunderbird*** alkönyvtárban találjuk. Szerkesszük át a már ismert módon ezt a fájlt is úgy, hogy az immár a megosztott partíción található beállítási állományokra mutasson. Ehhez az `IsRelative` jelző értékét ismételten nullára kell állítanunk, a ***Path***-ban pedig megadni a ***/share*** alatti fájlrendszeren található könyvtár teljes elérési útvonalát.
Az én módosított linuxos ***Thunderbird*** beállításaim tehát a következőképpen néztek ki:

```
[General]
StartWithLastProfile=1
[Profile0]
Name=default
IsRelative=0
Path=/share/users/kevin/Thunder
➥bird/Profiles/ 19ximc3t.default
Default=1
```

Ha készen vagyunk, indítsuk el a ***Thunderbird*** programot, és nézzük meg, hogy láthatóak-e ugyanazok a levelek és postafiókok, amelyeket eddig ***Windows*** alatt használtunk. Ha a program új profilt akar létrehozni, az megint rossz jel. Ilyenkor a korábban leírtaknak megfelelően lépjünk ki belőle, és keressük meg a hibát.

Összefoglalás

Sokszor bizony kifejezetten kényelmes, ha az ember hordozható számítógépén ***Linux*** és ***Windows*** egyaránt van. Ezt a kényelmet akár fokozni is lehet azzal, ha a mindkét operációs rendszer alatt megtalálható alkalmazások beállításait és adatait megosztjuk a két rendszer között. Ha például úgy tudjuk futtatni a ***Mozilla Firefoxot*** és a ***Thunderbirdöt***, hogy közben nem kell állandóan arra figyelnünk, melyik rendszerben is vagyunk, az felbecsülhetetlen.
Bár egy ilyen megosztott konfiguráció kialakításához számos módosítást kell elvégezünk, maga a dolog végső soron nem bonyolult, különösen olyasvalaki számára, akinek sem ***Linux*** sem ***Windows*** alatta nem jelent problémát néhány fájl megkeresése, átmásolása és minimális átszerkesztése, vagy egy új könyvtárszerkezet kialakítása.

Linux Journal 2006., 146. szám

Kevin Farham elsősorban szoftverfejlesztéssel kapcsolatos projekteken dolgozik. Ezek között egyaránt akad dokumentumok indexelésével, matematikai modellezéssel és szimulációval, tudományos adatgyűjtéssel, adatelemezéssel és megjelenítéssel kapcsolatos munka. Cége a Lyra Technical Systems, Inc. Connecticut állam északkeleti részén található.

KAPCSOLÓDÓ CÍMEK

➲ www.linuxjournal.com/article/8954

Synaptic,
a csomagtelepítők gyöngye

Csomagokat telepíteni manapság már gyerekjáték. Hatalmas a választék, folyamatosan bővül a kínálat. Persze ez nem mindig volt így, ehhez az is kellett, hogy a Linux, mint desktop operációs rendszer is megállja a helyét. Ebben a cikkben egy grafikus csomagtelepítővel fogunk megismerkedni, ami GTK-s felületével és apt-t használó „motorjával" méltán lehet a kedvencünk.

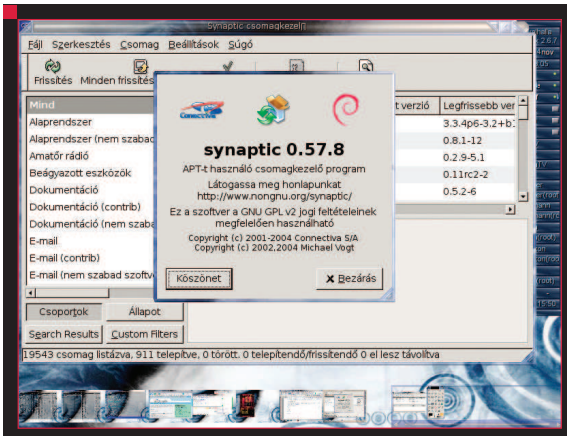
Visszaemlékezve, nekem mindig is az ***apt*** volt „a csomagtelepítő". Ugyan a ***linuxos*** karriéremet 2000-ben kezdtem ***Redhat 5.2-vel***, majd 2002 környékén komolyabban kezdtem foglalkozni a ***Mandrake 8.1*** és a ***SuSE 7.2*** terjesztésekkel, de ezekkel csak ímmel-ámmal foglalkoztam.
Az első igazán szimpatikus disztribúció nekem az ***UHU-Linux 1.0*** volt. Tudom, az ***UHU-t*** már sok cikkemben megemlítettem, de muszáj, ha hiteles akarok lenni. Ha nincs ez a rendszer, akkor ma biztosan nem ***Debian GNU/Linux-ot*** használnék. Szóval visszatérve az ***UHU-ra***, rögtön szimpatikus lett a csomagkezelése. Persze akkoriban fogalmam sem volt az ***apt*** és az ***rpm*** tényleges különbségeiről, illetve a hozzájuk kapcsolódó specifikus csomagtelepítőkről, de valahogy mindent nagyon leegyszerűsített az, hogy egy paranccsal letölthetek, majd telepíthetek programokat a számítógépemre. Tehát beleszerettem az ***apt-be***, mely szerelem a mai napig tart. Időközben azért megismertem más hasonló elven működő telepítőket, de az ***apt*** nálam toronymagasan vezet. Ez is, mint minden, megszokás kérdése, mivel a ***Linuxban*** az a jó, hogy rengeteg alternatívát kínál nekünk. Ebben a cikkben egy ***apt-re*** épülő grafikus ***frontend-del*** (felülettel), a ***Synaptic-kal*** fogunk foglalkozni.

Csomag

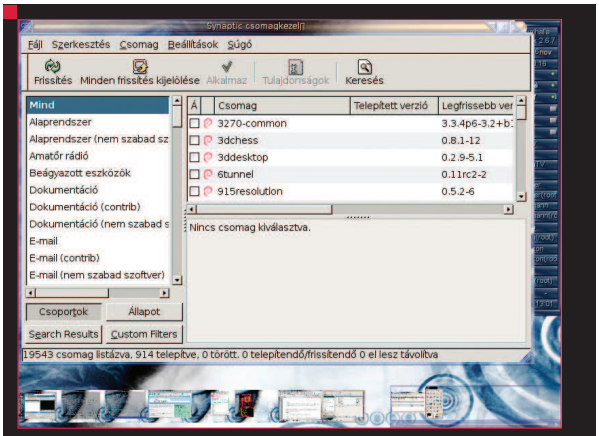
Nos, igen. Egy kezdő felhasználóban bizony felmerülhet a kérdés, hogy miféle csomagról is beszélek én folyton? Igen, közeleg a karácsony, de azért senki se gondolja, hogy egy linuxos szakmai lapban ajándékcsomagokról írok. Jelen esetben a csomag (***package***) magát a programot jelenti, amit telepíteni szeretnénk. Persze ez így eléggé konyhanyelvi fogalmazás, de ez a lényege. Egy bináris csomag tartalmazza a már lefordított forráskódot (kvázi magát a programot, innen kapta a mi csomagunk az angol ***binary package*** nevet), információkat (hova kell telepíteni, mi a verziószám, stb.), illetve magát a telepítő scriptet, ami elvégzi a telepítést. Ha most újra kiszaladnék a konyhába, akkor ott biztosan azt mondanám, hogy egy ***linuxos*** csomag körülbelül megfelel a ***windowsos setup.exe-nek***. Természetesen itt nem ér véget a sor. ***Linuxos*** csomag tartalmazhat képet, zenét, dokumentumokat, szóval mondhatni bármit.

Csomagformátumok

Ahány disztribúció, annyi formátum – mondhatnánk. Na azért ez nincs teljesen így, de elég közel áll az igazsághoz. Két fő formátumról (***linuxos*** csomag kiterjesztése) írnék pár szót. Az egyik a ***.deb*** kiterjesztésű csomagok, melyek a ***Debian GNU/Linux-ról*** kapták a nevüket. Természetesen nem csak ***Debian*** alatt használatos a ***.deb***, hanem majdnem minden ***dpkg-alapú*** (lásd később) terjesztésnél, például az ***Ubuntunál*** is.
A másik nagyon népszerű formátum a ***.rpm***, amit legelőször ***Redhat Linux*** alatt használtak (jó régen). Rengeteg mai „felkapott" disztribúció is használja az ***rpm-et***, mint például a ***SuSE***, a ***Mandriva*** és még sorolhatnám.



1. ábra A Synaptic egy apt-re szakosodott frontend



2. ábra A Synaptic felülete

Csomagkezelők

Több fajta csomagkezelő létezik. A csomagkezelők, mint a nevük is mutatja, a ***linuxos*** csomagok kezelésére irt programok, melyekkel telepíthetünk, listázhatunk, eltávolíthatunk, stb. csomagokat. Ez a cikk most nem kifejezetten erről szól, úgyhogy mélyen nem mennék bele a dolgok sűrűjébe, viszont azért nagy vonalakban essék szó ezekről. Maguk a ***Linux*** disztribúciók is csomagkezelő alapján csoportosíthatók. Három fő csoport van, ezek a ***dpkg***, az ***rpm***, illetve a ***tgz***.

A ***DPKG*** (***Debian Package***) a ***Debian GNU/Linux*** által lett világhírű, de természetesen más ***dpkg-alapú*** terjesztések is fellelhetők, mint például a már említett ***UHU-Linux***. Könnyű a kezelése, bárki könnyen elsajátíthatja.

Az ***RPM*** (***Redhat Package Manager***) a ***Redhat Linux*** csomagkezelője. Több dologban eltér a ***dpkg-től***, viszont ennek a használata is egyszerű. A legismertebb ***rpm-alapú*** disztribúciók a ***SuSE***, a ***Fedora*** és a ***Mandriva***.

A ***TGZ*** csomagkezelő (ami valójában nem is a csomagkezelő, hanem a csomag neve) a ***.tar.gz*** kiterjesztésről kapta a nevét, ami általában az összecsomagolt források kiterjesztése. Ez a csomagkezelő sok mindenben egyszerűbb, letisztultabb, mint a ***DPKG*** és az ***RPM***. A legfőbb ***tgz-alapú*** disztribúció a ***Slackware Linux***, ami az egyik legrégebbi (és a mai napig is fejlesztett) terjesztés. Persze számtalan más disztribúció is használja a ***tgz-t***, ahogy ezt már ***linuxos*** berkekben megszokhattuk.

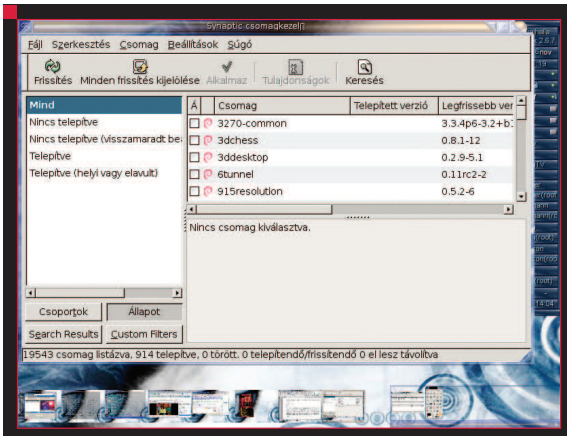
Csomagtelepítők

Bajban vagyok ennek a résznek az alcímével, mert pontosan nem tudom, hogy hogyan is lehetne nevezni azt az alkalmazást, amiről írni készülök. Mindenesetre én csomagtelepítőnek neveztem el, de ennél jóval több és jóval bonyolultabb. A legfőbb erénye mégis a telepítés a többi velejárójával együtt (eltávolítás, stb.). A leglényegesebb különbség az általam csomagtelepítőknek nevezett program és a csomagkezelők között, hogy ez le is tölti a csomagokat az internetről, nem beszélve a függőségek automatikus kezeléséről és a függőségi csomagok letöltéséről.

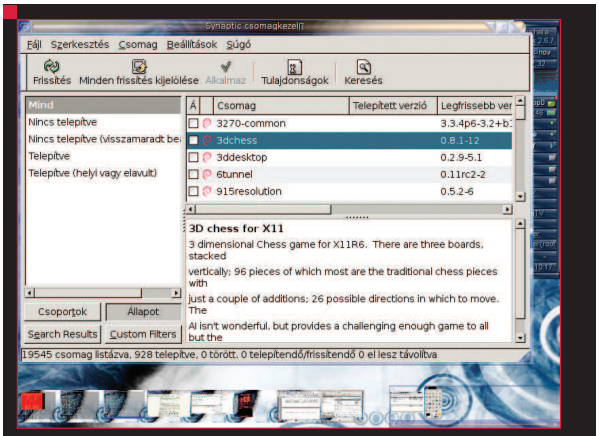
Bizony, itt egy újabb fontos területre tévedtünk az egész „csomagkezelésmizériával". Mégpedig a csomagoknak vannak függőségei. Hogy mit is jelent ez? Képzeljünk el egy piramis-szintű hierarchiát, mint a történelemórákon. Viszont itt nem ókori népcsoportok, királyok és rabszolgák szerepelnek az egyes „fakkokban", hanem csomagok. Tehát a csomagok egymásra épülnek. Az egyiknek szüksége van a másikra ahhoz, hogy tökéletesen funkcionáljon. Hogy ne eresszem túlságosan bő lére a mondandómat, nézzük meg világosan és érthetően (hogy azt ne mondjam ismét: konyhanyelven), hogy mi a csuda is a csomagtelepítő. Képzeljünk el egy olyan programot, ami egy bizonyos adatbázist használva, az internetről letölti a kívánt csomagot (a telepítendő programot), majd a csomagkezelő segítségével ezt előkonfigurálja, tehát elvégzi a szükséges beállításokat és ezt követően telepíti is. De ez még mind semmi, mivel már a letöltés előtt elvégzi a szükséges függőségek bírálatát és a függőségi csomagokat is letöltésre kínálja. Mindezt egy darab parancs segítségével. Nyilván annak, aki már használt ilyesmit, nem mondtam újat, de mivel az újszülöttnek (most a kezdőbb felhasználónak) minden vicc új, ezért lássunk erre egy példát:

```
root@valhalla:/home/rhiad#
➥apt-get install mozilla
Csomaglisták olvasása... Kész
Függőségi fa építése... Kész
Az alábbi extra csomagok
➥kerülnek telepítésre:
  mozilla-browser mozilla-
➥mailnews mozilla-psm
Javasolt csomagok:
  mozilla-chatzilla xprt latex-
➥xft-fonts
Az alábbi ÚJ csomagok lesznek
➥telepítve:
  mozilla mozilla-browser
➥mozilla-mailnews mozilla-psm
0 frissített, 4 újonnan telepí-
➥tett, 0 eltávolítandó és 0
➥nem frissített.
Letöltés az archívumokból:
➥11,7MB
Kicsomagolás után 35,2MB
➥lemezterületet használok fel
Folytatni akarod [Y/n]?
```

Elemezzük ki ezt a néhány sort. A feltevés szerint a ***Mozilla*** böngészőt szeretnénk telepíteni, ehhez beírjuk a megfelelő parancsot. A program megkeresi az adatbázisban a ***Mozilla-t***, utána megnézi, hogy milyen függőségei vannak. Ezek után



3. ábra Csomagok állapota



4. ábra Csomaginformációk

kilistázza szelektíven a telepítendő csomagok neveit aszerint, hogy mik az extra csomagok (függőségek), mik a javasolt csomagok (ezeket nem muszáj telepíteni). Végül kiírja az összes telepítendő csomag nevét, illetve a fájlok méretét. Ha a kérdésre igennel (***y*** vagy ***enter***) válaszolunk, elkezdődik a letöltés:

```
Folytatni akarod [Y/n]? y
Letöltés:1
➥ftp://ftp.de.debian.org
➥unstable/main mozilla-browser
➥2:1.7.13-0.3 [9719kB]
Letöltés:2
➥ftp://ftp.de.debian.org
➥unstable/main mozilla-
➥mailnews 2:1.7.13-0.3
➥[1787kB]
Letöltés:3
➥ftp://ftp.de.debian.org
➥unstable/main mozilla-psm
➥2:1.7.13-0.3 [187kB]
Letöltés:4
➥ftp://ftp.de.debian.org
➥unstable/main mozilla
➥2:1.7.13-0.3 [1030B]
Letöltve 11,7MB 14m43s alatt
➥(13,2kB/s)
Csomagok előkonfigurálása ...
Új csomag kiválasztása:
➥mozilla-browser.
(Adatbázis olvasása ... Most
➥68181 fájl és könyvtár
➥telepített.)
Kicsomagolás: mozilla-browser
➥innen: .../mozilla-
➥browser_2%3a1.7.13-
➥0.3_i386.deb ...
Új csomag kiválasztása:
➥mozilla-mailnews.
Kicsomagolás: mozilla-mailnews
➥innen: .../mozilla-
➥mailnews_2%3a1.7.13-
➥0.3_i386.deb ...
Új csomag kiválasztása:
➥mozilla-psm.
Kicsomagolás: mozilla-psm
➥innen: .../mozilla-
➥psm_2%3a1.7.13-0.3_i386.deb
➥...
Új csomag kiválasztása:
➥mozilla.
Kicsomagolás: mozilla innen:
➥.../mozilla_2%3a1.7.13-
➥0.3_i386.deb ...
Beállítás: mozilla-browser
➥(1.7.13-0.3) ...
Updating mozilla chrome
➥registry...done.

Beállítás: mozilla-mailnews
➥(1.7.13-0.3) ...
Updating mozilla chrome
➥registry...done.

Beállítás: mozilla-psm
➥(1.7.13-0.3) ...
Updating mozilla chrome
➥registry...done.

Beállítás: mozilla (1.7.13-0.3)
➥...
```

Ráadásul szépen fel is települ a böngésző. Számos ***apt-hez*** hasonló alkalmazás létezik. Például az ***urpmi*** az ***rpm-alapú*** rendszerekhez, a ***swaret*** vagy ***slapt-get*** (***apt-klón***) a ***tgz-alapú Slackware-hez***, vagy az ***emerge Gentoo-hoz***. Most egy kicsit behatóbban fogunk foglalkozni az ***apt-vel***, mivel a ***Synaptic*** erre épül.

Az apt

Az ***Advanced Packaging Tool*** (***APT***) legelőször a ***dpkg-hez*** jelent meg, mint egy bő egyéb szolgáltatásokat nyújtó ***frontend***. Mint már említettem, az ***apt*** képes a csomagokat letölteni és telepíteni az

```
apt-get install csomagnév
```

paranccsal, de ugyanígy el is távolíthatunk nem kívánt csomagokat az

```
apt-get remove csomagnév
```

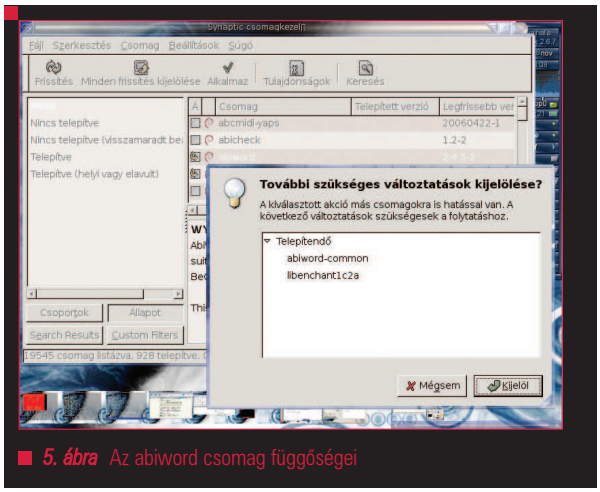
segítségével. Fontos megjegyeznem, hogy egyszerre természetesen több csomagot is telepíthetünk vagy távolíthatunk el. Ekkor a parancsok így módosulnak:

```
apt-get install csomagnév1
➥csomagnév2 ... csomagnévn
```

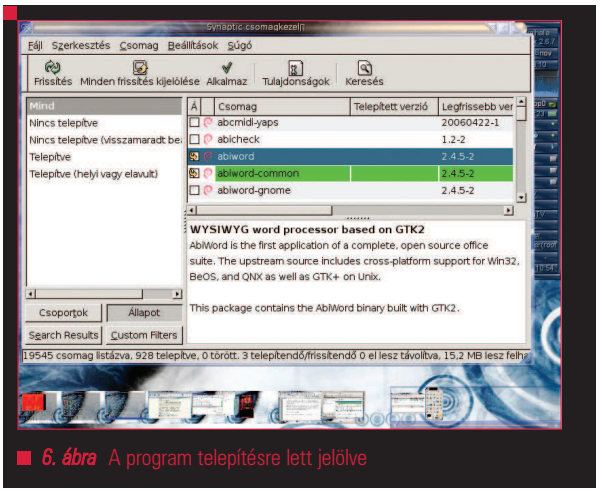
és

```
apt-get remove csomagnév1
➥csomagnév2 … csomagnévn
```

Nyilván ezek a parancsok mit sem érnek az ***apt*** beállítása nélkül. Ez általában gyárilag már minden rendszerben helyesen be van állítva, de az esetek többségében (ha egy kicsit már ismerjük a rendszert) szükség van utólagos „hegesztésre". Az ***apt*** egy listában tárolja a „csomaglelőhelyeket", a /etc/apt/sources.list fájlban.
Az ***ftp-*** vagy ***webhelyeken*** persze a csomagok verziója mindig frissül, illetve kerülnek be új csomagok a kínálatba.



5. ábra Az abiword csomag függőségei



6. ábra A program telepítésre lett jelölve

Az ***apt*** naprakészségéhez minden telepítés előtt érdemes kiadni az

```
apt-get update
```

parancsot, mely frissíti az adatbázist. Az

```
apt-get upgrade
```

frissíti a rendszerben fellelhető összes csomagot, így naprakész rendszerünk lesz (persze csak azokat a csomagokat, melyeknek van frissebb verziója). Az

```
apt-get dist-upgrade
```

magát a rendszert frissíti. Ez a parancs rendszerszintű verzióváltásoknál szükséges.
Fontos megjegyeznem, hogy az ***apt*** a letöltött csomagokat telepítés után nem törli le, hanem elraktározza a */var/cache/apt/archives* könyvtárba. Fontos parancs lehet a kis ***rootpartícióval*** (vagy kicsi */var* partícióval) rendelkező felhasználóknak az

```
apt-get clean
```

parancs, mely arra hivatott, hogy az imént említett könyvtár tartalmát törölje.
Utolsóként essen még szó az

```
apt-cache search bármi
```

nevűről. Ez a parancs az információs adatbázisban keres bármire, amit megadunk neki. Tehát ha a `bármi` helyébe `browser`-t írunk, akkor kilistázza az összes olyan csomagot, aminek a nevében vagy a leírásában szerepel a ***browser*** szó.
Az ***apt*** parancsait mindig rendszergazdaként (azaz ***root-ként***) kell elvégezni, mivel egy sima felhasználónak nincs erre jogosultsága.
Természetesen az ***apt*** szolgáltatásainak csak töredékéről esett itt szó. Egy nagyon részletes és jó leírást találhatunk a ➲ ***http://people.inf.elte.hu/radicsla/Linux/irasok/apt-hogyan/apt_howto.hu.html*** oldalon, amennyiben még inkább szeretnénk megismerni ezt az eszközt.

Apt más rendszerekre

Az ***apt*** olyan jónak bizonyult az idők során, hogy rengeteg más disztribúció is „átvette” magának. Megjelent az ***apt4rpm***, ami lehetővé teszi az ***apt*** használatát az ***rpm-alapú*** rendszereken is. Amennyiben ***rpm-alapú*** terjesztésünk van és szeretnénk ***Synaptic-ot*** használni, látogassunk el a ➲ ***http://apt4rpm.sourceforge.net/*** weboldalra és töltsük le ezt a programot.

Grafikus csomagtelepítők

Eddig úgymond konzolon léteztünk a cikk folyamán, most evezzünk át grafikus vizekre. Tudniillik ezen műveleteket nem csak parancssorban lehet elvégezni, hanem grafikus programok segítségével is. Persze ezek nem egyéni programok, hanem egyes konzolos csomagtelepítőkre „húzott” ***skin-ek***, vagy ***frontend-ek***. Maradjunk annyiban, hogy megkönnyítik az életünket, a kezdőbb felhasználók életét meg még inkább.

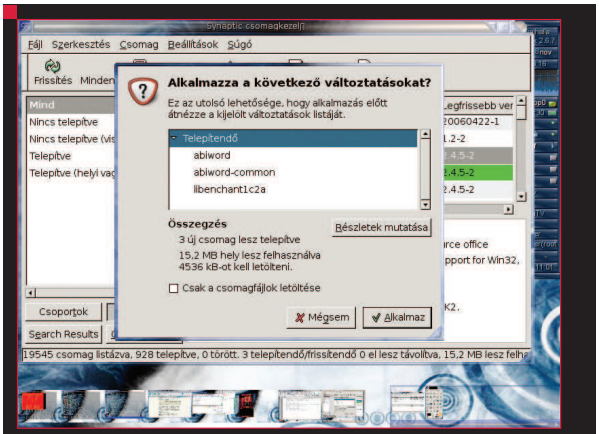
Gondoljunk csak a valóban kiváló ***Mandriva Vezérlőközpontra***, melynek a csomagtelepítője az ***urpmi-re*** épül. De mondhatnám akár a ***KPackage-et*** is, ami a ***KDE*** alapértelmezett csomagtelepítője. Egy másik alternatívát nyújt a ***Synaptic***.

Synaptic

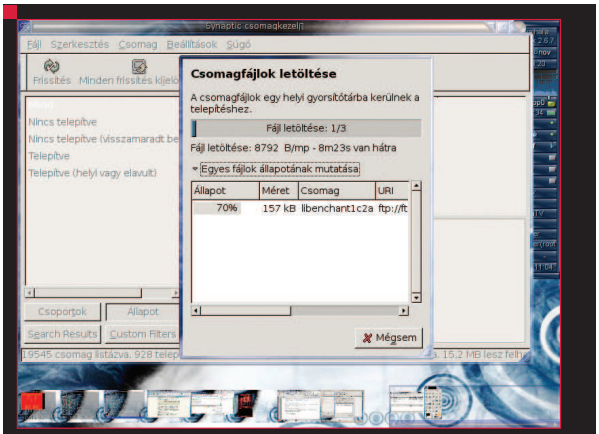
A ***Synaptic*** (➲ ***http://www.nongnu.org/synaptic/index.html***) egy ***GTK+-ra*** épülő grafikus frontend az ***apt-hez***. Képes elvégezni a csomagtelepítési funkciókat grafikus felületen keresztül, egér használatával, de természetesen, ahogy azt már megszokhattuk, minden fontos funkcióhoz kapcsolódik billentyűkombináció is. Számomra hatalmas előnye a parancssorral szemben, hogy itt a saját szemünkkel láthatjuk az összes csomagot kilistázva és azok közül válogathatunk. Tehát semmit sem kell tudni előzőleg egy csomagról, mint a konzolos ***apt-nél***.
Jelenlegi legfrissebb verziószáma a ***0.57.11***. Letölteni kétféle formában lehet ezt az alkalmazást: forrásból a ➲ ***http://savannah.nongnu.org/files/?group=synaptic*** oldalon érhető el, illetve ***.deb*** csomagban a ***Debian GNU/Linux*** mindhárom agában (***stable***, ***unstable***, ***testing***). Természetesen más disztribúciók is tartalmazzák a ***Synaptic-ot***, mint például a ***SuSE Linux***.

Telepítés

Amennyiben forrásból szeretnénk telepíteni, figyeljünk a függőségekre. Ezek a ***GTK+ 2.4*** (vagy ennél magasabb verzió) és természetesen az ***apt***.



7. ábra Utolsó figyelmeztetés



8. ábra Csomagok letöltése

Ha ***Debian GNU/Linux-ot*** használunk, akkor a telepítés kizárólag az

```
apt-get install synaptic
```

parancsból áll. Amint azt már az imént kitárgyaltuk, az ***apt*** mindent megcsinál helyettünk. Néhány disztribúció alapértelmezettként is tartalmazza a programot, például az ***UHU-Linux*** is ilyen. Ez esetben nincs szükségünk telepítésre.

Használat

Indítsuk el az alkalmazást menüből, vagy gépeljük be bármilyen ***xterminálba*** a

```
synaptic
```

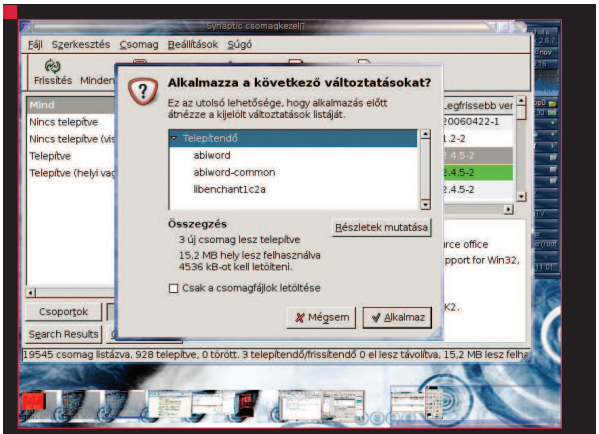
parancsot. Vigyázzunk arra, hogy ez az alkalmazás csak ***root-ként*** fut! Normál felhasználóként a

```
gksu synaptic
```

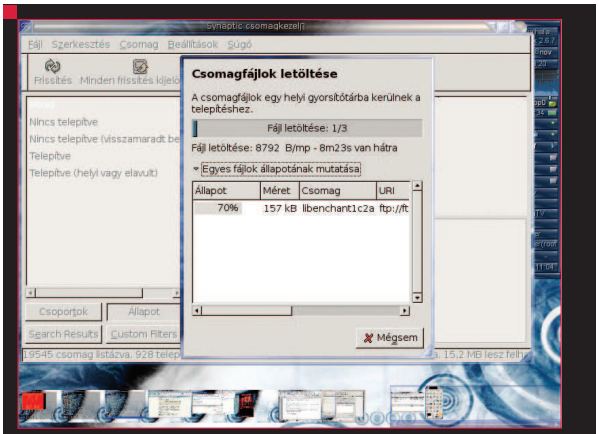
paranccsal indítható, de ehhez először fel kell telepítenünk a ***gksu*** (grafikus felületen rendszergazda-jogú programok futtatása sima felhasználóként) nevű programot. A ***rootjelszó*** beírása után megjelenik a képernyőnkön a ***Synaptic***.

Funkciók

Lássuk akkor, hogy miből is élünk! Végre a hosszúra nyúlt bevezető után elérkeztünk a lényegi részhez. Először is vegyük szemügyre magát a ***Synaptic*** ablakát. Aki olvasta az ***Evolution-ről*** szóló cikkemet (vagy ***Evolution-t*** használ), az észreveheti, hogy a ***Synaptic*** ablaka szakasztott olyan, mint az ***Evolution-é***.
Ez rögtön jó jel, mert biztosít minket a könnyű használatról és felhasználóbarátságról.
Vizsgáljuk meg az ablak bal oldali részét. Itt kategóriák vannak felsorolva a csomagok ilyen-olyan tulajdonságai szerint. Ezek a kategóriák is kategóriákba vannak sorolva, melyek között a bal alsó gombokkal választhatunk. Az első ilyen a ***„Csoportok”*** gomb (alaphelyzetben ez van kijelölve). Ebben a kategóriában a csomagok funkcióik alapján vannak csoportosítva. Például itt találhatjuk meg a grafikus felületeket, játékokat, fejlesztőeszközöket, és így tovább.
A második kategória a fellelhető csomagok állapotbeli besorolását mutatja. Ez lehet az összes csomag, a telepített csomagok, vagy a még nem telepített csomagok. Hasznos ez az összetétel, hogyha például csak a nem telepített csomagok között böngészünk. Így megtudhatjuk, hogy még mink nincs.
A harmadik és a negyedik gomb (alsó kettő) a keresési eredményekre van kihegyezve.
Az ablak legnagyobb részét az a terület foglalja el, ahol maguk a csomagok sorakoznak szép ábécésorrendben. Ezt a sorrendet persze lehet másképp is felállítani, például verziószám, vagy fájlméret alapján.
Egy csomagot tartalmazó sor a következőképp néz ki: állapotjelző, ikon (nálam Debian jel), csomagnév, telepített verzió száma, legfrissebb verzió száma, fájl mérete, egy mondatos információ.
A csomagnevek alatt az éppen kijelölt csomagról találunk bőséges információt.
Ez általában a kiválasztott csomag funkcióit írja le részletesen. A felület legalján található állapotsor pedig az aktuális szekcióról ad információt, illetve általánosabb témákat is megvilágít. Például az összes csomag és az összes telepített csomag számát is leolvashatjuk róla.
A ***Synaptic*** majdnem az összes ***apt*** parancsot „tudja”, beleértve a `dist-upgrade`-et is. Tartalmaz egy remek keresőt, ami az `apt-cache search`-nek felel meg. Mivel az ***apt-nél*** már tárgyaltuk az egyes parancsokat, ezért újra nem térnék itt ki rájuk. Minden nagyon világosan látszik a menükből, tényleg gyerekjáték a kezelése.
Nézzük meg gyakorlatban ezt az alkalmazást! Legyen az „áldozat” az ***abiword*** nevű népszerű pehelysúlyú szövegszerkesztő. Tehát ezt szeretnénk feltelepíteni.
Legelőször telepítésre kell jelölnünk. Ezt a legegyszerűbben úgy tehetjük meg, ha kétszer kattintunk a nevére. Természetesen, mint minden akciót, ezt is lehet menüből végezni.
Amennyiben kijelöltük, a csomag neve melletti kis négyzetben megjelenik egy nyíl.
Mint az ***apt***, a ***Synaptic*** is rögtön megmutatja egy csomag függőségeit. Ezek a csomagok szükségesek az ***abiword*** telepítéséhez és futtatásához, tehát el kell fogadnunk őket. Ha elfogadtuk, a függőségek is telepítésre lesznek jelölve. Most már nincs más dolgunk, mint az ***„Alkalmaz”*** ikonra kattintani az ikonsorban. Ekkor



9. ábra Csomagok telepítése



10. ábra A telepítés sikeresen befejeződött

kapunk egy utolsó figyelmeztetéssel egybekötött összegzést. Ezt jóváhagyva elkezdődik a csomagok letöltése, majd a telepítés. Végül, ha minden jól ment, kapunk egy üzenetet, hogy minden csomag sikeresen a helyére került. Az ***apt*** kimenetét végig figyelemmel kísérhetjük a procedúra során. Ehhez a *„Details”* nevű lenyíló menüre kell kattintanunk. Az eltávolítás és az egyéb ***apt-műveletek*** hasonlóan történnek, mint a telepítés.

Remélem sikerült felkeltenem mindenki érdeklődését a ***Synaptic*** és az ***apt*** iránt. Remek kis találmány mindkettő, megkönnyíti az életünket. Frissen tarthatjuk vele rendszerünket és láthatjuk, hogy pontosan milyen programokat is használunk. Én személy szerint a parancssort részesítem előnyben, de néha azért „előkapom” a ***Synaptic-ot*** is.
Kellemes csomagtelepítést mindenkinek!

Apagyi György, (killall)
(killall@root.hu)

25 éves, jelenleg az ELTE programozó matematikus szakán másodéves hallgató. Hobbija a zene (gitározás), az olvasás (Stephen King) és a számítástechnika (Linux, Unix, VMS).

Vékony kliensekkel jobban megéri

Költséghatékony, merevlemez nélküli vékony kliensek az oktatásban és a munkahelyeken.

Az *Esteliben*, *Dél-Nicaraguában* folyó *Superemos* közösségi oktatási programban *Linux* kiszolgálóval működtetett használt gépekből kialakított, merevlemez nélküli vékony klienseket vezettünk be. Hasonló rendszereket sok más szervezetnél is alkalmaznak. Ezzel pénz takarítható meg, ugyanakkor egyszerűbbé teszi a rendszeradminisztrációt és fokozza a biztonságot, így ideális megoldás lehet olyan alacsony költségvetésű oktatási programok számára, mint az említett nicaraguai. Az interneten számos leírás és dokumentáció elérhető a témában, mégis, a megvalósítás bonyolultnak tűnhet. A befektetett munka azonban biztosan megtérül. Ebben a cikkben bemutatjuk, hogyan építettük fel a hálózatunkat a rendszerindításhoz régi merevlemezeket és *Flash* meghajtókat használó vékony kliensekből. A leírás hasznos lehet mindenkinek, aki alacsony költségek mellett akar nagyszámú ügyfélnek hozzáférést biztosítani számítástechnikai erőforrásokhoz.
A projektünk sikerességét az is mutatja, hogy az oktatók hamar megértették, hogy a régi, használt számítógépek újrahasznosíthatók és akár a legújabb szoftverek futtatására is képessé tehetők. Hozzá kell tenni, egy hasonló megoldásból jobban finanszírozott szervezetek, üzleti vállalkozások és kormányzati hivatalok is profitálhatnak.
Mielőtt bemutatnánk a lemez nélküli kliensekből álló hálózatok kiépítésnek alapjait, előbb ejtsünk pár szót a projektünkben felhasznált forrásokról és eszközökről. A különböző régi és új gépeken a Novell *Suse Linuxának 10.0* változatát és az *Ubuntu Breezy Badger*

kódnevű terjesztését használjuk operációs rendszerként. Gépeket és alkatrészeket a *Rotary Club of Toronto-Leaside* és a *Linux Journal*t kiadó *SSC Inc.* biztosítottak. Meg kell említenünk, hogy az összes gép *PC*. Érdemes egyébként egységesíteni a felhasznált eszközöket, bár ez általában igen nehéz, ha öröklött és adományozott eszközökkel kell dolgozni.
A projektünket működtető eszközök között van még a *Linux Terminal Server Project* vékony kliensekhez fejlesztett kitűnő szoftvere és átfogó, *boot-ROM* képfájlokat tartalmazó könyvtára (lásd az online forrásokat). A program, amellyel felkészítettük a régi merevlemezeket a *boot-ROM* helyettesítésére *Andy Rabagliati* munkája.

A lemez nélküli kliens hálózatok alapjai

A lemez nélküli ügyfelekből álló hálózathoz is szükségeltetik egy kiszolgáló. Esetünkben ezt *Ethernet* kábellel kötöttük a hálózatba.
Az ügyfelek erről a kiszolgálóról kapják meg az operációs rendszerüket. Amint egy ügyfélgép elindul, a *BIOS* alapján rájön, hogy a merevlemezén nincs operációs rendszer, így megpróbál hozzájutni egyhez a helyi hálózaton (*LAN*) keresztül, egy kérést küldve a kiszolgálónak. A kiszolgáló fogadja a kérést, majd megnézi, hogy van-e megfelelő operációs rendszere, amelyet visszaküldhet. Ha van, az ügyfél a szokásos módon betölti azt, a felhasználó pedig észre sem veszi, hogy gépének operációs rendszere nem a sajátja, hanem a hálózatról származik.
Eltartott egy ideig, mire megértettük az egész dolog működését. Az első lépés, hogy kideríjtük, az ügyfélgép *BIOS*-a tartalmaz-e olyan beállításokat, amelyek lehetővé teszik, hogy a hálózatról induljon (*„Boot from LAN" – indítás helyi hálózatról*) *boot-ROM*-on keresztül. Némelyik gépen ez egyértelmű, másokon a beállítás további alopciókkal történik, megint másokon nincs is ilyen lehetőség. Ha van, az azt jelenti, hogy a gép képes *boot-ROM* chip és gyakran a hálózati kártyán található *Pre-boot Execution Environment (PXE)* segítségével indulni.
Ha megtaláltuk a megfelelő opciót, beállítottuk és még szerencsések is vagyunk, akkor működni fog a dolog Ha mégsem, nem kell kétségbe esni. Egyik *VIA* chipkészletes gépünk azt állította, tud hálózatról indulni

PXE-t használva, aztán mégsem indult. Aztán egyik nap hirtelen mégis. A hasonló problémák a lemez nélküli ügyfél hálózatok felépítésének velejárói, de túl kell rajtuk esnünk, hogy végül egy elsőosztályú vékony kliens hálózatot kapjunk, amelybe bármilyen gépet bekapcsolhatunk.

Némely gép az alaplapra integrált hálózati kártyával rendelkezik. Ha nem, akkor többnyire egy *PCI* csatlakozóban helyezkedik el a hálózati kártya. (Használhatunk persze régebbi *ISA* kártyákat is, de azok további beállításokat igényelnek, így inkább kerültük őket) Ha a kártya nem integrált, valószínűleg nem rendelkezik előre telepített *boot-ROM*-mal sem.

A projekt során két komolyabb problémával találkoztunk. Az egyik, mikor a leendő ügyfélgép *BIOS*-a nem kínál lehetőséget a hálózatól való indításra. A második, mikor kínál, ám a hálózati kártyának nincsen *boot-ROM*-ja. Mindkét problémát sikerült azonban áthidalni úgy, hogy egy régi merevlemezen, vagy *USB Flash* meghajtón elhelyeztünk olyan fájlokat, amelyek *boot-ROM*-ot imitálnak. A cikk nagy részét ennek bemutatása fogja kitenni. A megoldáshoz nincs szükség floppy lemezekre, amelyek egyébként itt Nicaraguában nem is működnek túl megbízhatóan.

Az ügyfélgépek 32 MB RAM-mal már működni fognak, bár jobban érzik magukat 64 MB-tal. Régebbi gépek, 266 MHz körüli processzorral tökéletesen megteszik, de nyilván a gyorsabbak jobban működnek. A kiszolgálón beállíthatunk régebbi egereket, monitorokat és nem-angol billentyűzeteket, ha szükséges. A kitűnő *LTSP* szoftvernek köszönhetően a legtöbb eszköz nem igényel külön konfigurációt.

Nagyobb befektetés csak a kiszolgálóhoz szükséges. A kiszolgálónkban most 1GB RAM és egy 2.4 GHz-es processzor működik, így egészen gyorsan tud kiszolgálni több, mint egy tucat ügyfelet, akik az interneten böngésznek, irodai alkalmazásokat és játékokat futtatnak. Egyetlen, megfelelően felszerelt kiszolgálóval akár több tucat ügyfelet is elláthatunk.

Itt most nincs hely a kiszolgáló teljes beállításának bemutatására, ám ezt már úgyis megtette *Kevin Brown* (*lásd a kapcsolódó címeket*).

A szükséges fájlok beállítása

A projektünkhöz a *LILO (Linux Loader)* rendszertöltőt használjuk, mivel úgyis csak *Linuxot* akarunk indítani. Fontos viszont a változatszám. Észrevettük, hogy a legújabb változat *lba32*-t használ a lemez geometriájának vezérlésére, ez pedig *Flash* meghajtóknál gondot okozott. Szerencsére a régebbi változatokkal nincs ilyen probléma. A *Flash* meghajtóinkon az *Andy Rabagliati wizzy* csomagjában található változatot használtuk fel, a régi merevlemezekhez pedig a *SUSE*-hoz és az *Ubuntuhoz* csomagolt változatokat. (Lásd a Régi merevlemezek beállítása részt, ahol a *GRUB* rendszertöltőt is megemlítjük.)

Ahhoz, hogy a *boot-ROM* lemezeink működjenek, szükségesek a különböző hálózati kártyák *boot-ROM* képfájljai is. A hálózati kártyáink *3Com 905*, *Realtek 8139*, vagy *Via-Rhine* típusúak. A képfájlokat a *Rom-o-matic*-ról szereztük. Kellett némi próbálkozás, míg megtaláltuk azokat, amelyek működnek. A *Rom-o-matic* rendszeresen frissíti a kiadásait, ezek a kiadások pedig hasonló opciókkal rendelkeznek. Mindegyiknél található egy gomb, mellyel megtekinthető a kompatibilis kártyák listája, hogy minimálisra csökkentsék a szükséges próbálgatások számát.

Ha megtaláltuk a megfelelő képfájlt, ki kell választanunk annak típusát. Mivel *LILO*-t használunk, ezért *zlilo*, illetve a régebbi *lzlilo* típust választottuk. Az *ltlilót* használtuk fel a *Flash* meghajtókon, mivel az újabb *zlilo* képfájlok csak merevlemezeken akartak működni. Rá akartunk viszont jönni arra is, hogy miért.

A kísérletezés persze nem vezetett eredményre, így most csak az eredményeket tesszük közzé. Mások, más eszközökkel bizonyára más és valószínűleg jobb eredményekre jutnak majd.

A *ROM-o-matic*-on egy *Get Rom* gomb megnyomásával tölthetjük le a képfájlokat. Megnyomására megjelenik egy dialógusablak a fájl mentéséhez a helyi fájlrendszerbe. Letöltöttük a megfelelő *.lzlilo* és *.zlilo* kiterjesztésű fájlokat a háromféle hálózati kártyánkhoz. Ezekkel és a *LILO* fájlokkal minden megvolt a *boot-ROM*-ok létrehozásához merevlemezen és *Flash* meghajtón egyaránt. Egyetlen könyvtárba másoltuk őket, amit */flashlilo*-nak neveztünk el. Ezek kerülnek majd a *boot-ROM* lemezünkre.

Telepítés Flash meghajtóval

Újabb gépek esetén, amelyek hálózatról nem indíthatók, de *USB* eszközről igen, a *Flash* meghajtó praktikusabb megoldás mint egy régi merevlemez. Amint a gép beindult, a *Flash* meghajtót kivehetjük és indíthatunk vele egy másik gépet. Rájöttünk, hogy a *Flash* meghajtó *SCSI* eszközként kezelhető, a *hotplugnak* köszönhetően pedig akár működés közben behelyezhető és eltávolítható. *Suse Linuxon* a *YaST* beállítóprogram rögtön felismerte az új eszközt és rákérdezett, hogy beállítsa-e. Azt mondtuk, ne.

Hogy ne ütközzünk formázási és particionálási problémákba, rendszergazdaként töröltük a meglevő partíciót, és létrehoztunk egy új indítható partíciót. (Ha van valami a lemezen, érdemes másolatot készíteni róla, mivel particionáláskor minden adat elvész). A *Flash* meghajtókat az alábbi módon particionáltuk:

```
# fdisk /dev/sda
```

A rendszertöltő beállításakor szükség lesz a fejek, szektorok és cilinderek számára – az `fdisk` mindegyiket kijelzi. Ne felejtsük el feljegyezni ezeket az értékeket. Az indítható partíció létrehozása bizonyára probléma nélkül lezajlik, ezután már csak a fájlrendszert kell létrehoznunk rajta. A legegyszerűbben ezt így tehetjük meg:

```
# mke2fs /dev/sda1
```

Így létrejött egy *ext2* fájlrendszer a *Flash* meghajtón. Ezután becsatoljuk azt a hagyományos */mnt* könyvtárba, persze csak miután ellenőriztük, hogy az üres:

```
# mount /dev/sda1 /mnt
```

A */flashlilo* könyvtár tartalmát átmásoljuk a */mnt*-be:

```
# cp /flashlilo/* /mnt
```

Ezután készítünk egy beállításfájlt a *LILO* számára. A *vi*-től és az *emacs*-tól ijedten a *pico* szerkesztőprogram mellett döntünk:

```
boot = /dev/sda
disk = /dev/sda
   bios = 0x80
   sectors = 62
   heads = 4
   cylinders = 1015
install = /mnt/boot.b
map = /mnt/map
root = /dev/sda1
vga = normal
read-only
delay = 30
pROMpt
image =
/mnt/viarhine6102.lzlilo
        label=viarhine2
        read-only
image = /mnt/3c905b.lzlilo
        label=3Com905b
        read-only
image = /mnt/rt8139.lzlilo
        label=RTL8139
        read-only
```

A szabványos ***lilo.conf*** néven mentjük a fájlt a ***/mnt*** könyvtárba.
A beállítások többsége a rendszerindítás folyamatának azt a részét érinti, ami a menü megjelenése előtt megy végbe. Az első sor utasítja a gépet, hogy a ***Flash*** meghajtóról induljon. A második sor és az alá tartozó sorok megadják a lemez geometriáját – ez a rendszertöltő helyének meghatározásához szükséges. (Itt használjuk fel az `fdisk` által kijelzett adatokat.) Az `install` sor megadja a ***boot.b***-t a rendszertöltő alapok telepítéséhez.
A `map` sor megmondja a vékony kliensnek, hogy hol találhatók a ***LILO*** által telepített állományok, a `root` pedig meghatározza a fájlrendszer helyét. A `vga` sor az adatok monitoron való megjelenítését szabályozza. A `read-only` opció hatására a gyökér fájlrendszer csak olvasható módban lesz becsatolva. A `delay` értéke a `prompt` megjelenítése előtti várakozási idő.
Az `image` részeknek köszönhetően az ügyfél különböző opciók közül választhat. A gép indulásakor a ***LILO*** egy menü formájában felajánlja ezeket a lehetőségeket, ez esetben, hogy a háromféle hálózati kártya közül – ***Via-RhineII***, ***Realtek 8139***, vagy ***3Com905*** – melyikről induljon.

Miért hivatkoztunk az ***/mnt*** könyvtárban levő ***boot.b***-re, a ***map***-re és a ***LILO*** képfájlokra? Most elkészültünk a beállításfájllal, így meg kell mondanunk a ***LILO***-nak, hogy azt használja.
Ezt csak ugyanabban a könyvtárban tehetjük meg, amelyikben dolgozunk, vagyis ahova a meghajtót csatlakoztattuk. Esetünkben ez az ***/mnt***.
A használandó ***LILO*** változatnak a következő paranccsal adhatjuk meg a beállításfájlt:

```
# /mnt/lilo -C lilo.conf
```

Ez biztosan jónak tűnik, amíg az ***/mnt*** könyvtárban vagyunk. De mi történik, ha lecsatoljuk a ***Flash*** meghajtót és áttesszük abba a gépbe, amelyet indítani akarunk vele? Nem kell megváltoztatni a ***lilo.conf*** hivatkozásait? És nem kell utána újra és újra futtatnunk a ***LILO***-t a megváltozott ***lilo.conf*** fájllal? Nem. Mikor megpróbáltuk így elindítani az egyik ügyfél gépünket, remekül működött. Tehát készen van a ***boot-ROM*** meghajtónk, lépjünk ki az ***/mnt*** könyvtárból és adjuk ki a következő parancsot, mielőtt eltávolítanánk a meghajtót:

```
# umount /dev/sda1
```

Régi merevlemezek beállítása

Eleinte a régi merevlemezeknél azt az eljárást folytattuk le, amellyel a ***Flash*** meghajtókat is beállítottuk, hogy ***boot-ROM***-ot imitáljanak. Ehhez viszont néha módosítani kellett a jumperek (a lemezek oldalán levő apró csatlakozók) elrendezésén is. A legtöbb lemezen diagram mutatja, hogyan kell a jumpereket elrendezni a különböző beállításokhoz, amelyek név szerint a ***Master*** (elsődleges), ***Slave*** (másodlagos) és ***Chain Select***. A ***Chain Select*** esetén a ***BIOS*** dönti el, hogy a merevlemez elsődleges, vagy másodlagos legyen.
A legtöbb ***PC***-ben egyetlen merevlemez van, ami többnyire az elsődleges. A ***Linux*** ezt ***/dev/hda***-ként ismeri fel. Ha a gépnek elég memóriája és elég gyors processzora van, érdemes csatlakoztatni egy ***CD-ROM***-ot is, jumpereit ***Chain Selectre***, vagy ***Slave***-re állítani, a merevlemezét pedig ***Masterre***. Az időmegtakarítás miatt aztán úgy döntöttünk, hogy egy újabb és gyorsabb, és újabb ***CD-ROM***-mal ellátott gépen végezzük el a merevlemezek előkészítését. Kivettük tehát az újabb gépből a merevlemezt, betettük az előkészítendő régit és ***Masterre*** állítottuk. Így miután visszaraktuk a vékony kliensbe és csatlakoztattuk az alaplaphoz, az rögtön elsődlegesként ismerte meg.
Bárhogyan készítsük is elő a régi merevlemezeket, utána telepíthetünk ***Ubuntu Breezy Badgert*** közvetlenül a ***CD***-ről, vagy ***SUSE 10***-et a helyi hálózatról. Minden esetben minimális, szöveges telepítést végezzünk és a ***LILO***-t válasszuk rendszertöltőnek. Amint kész a telepítés, megszerkeszthetjük az ***/etc/lilo.conf*** fájlt és hozzáadhatjuk a képfájlokat, ugyanabban a sorrendben, ahogy a ***Flash*** meghajtóhoz csináltuk, a korábban bemutatott ***lilo.conf***-ban, aztán frissíthetjük a ***LILO***-t az új beállításokkal:

```
# lilo -C /etc/lilo.conf
```

Volt egy régi lassú linuxos gépünk is amit vékony klienssé akartunk változtatni. Rájöttünk, hogy a rendszertöltőjének, a ***GRUB***-nak a menü fájljához elég hozzáadni az alábbi sorokat (helyettesítsük be a megfelelő képfájl nevét), hogy hajlandó legyen ***boot-ROM*** képfájllal indulni.

```
title Via-Rhine Boot-ROM
    root (hd0,0)
    kernel /boot/
    ➥via-rhine.zlilo
```

Az újraindításkor beállítottuk, hogy a merevlemezről induljon. A beállított gépben ugyanolyan hálózati kártya volt, mint némelyik régi gépünkben, amelyet vékony kliensnek akartunk beállítani, így jól jött, hogy megbizonyosodhattunk afelől, hogy a merevlemez remekül imitálja a ***boot-ROM***-ot. Ha minden működik, áttehetjük az újonnan beállított merevlemezt abba a gépbe, amelyet indítani fog. Ha úgy konfigurálunk egy régi gépet, hogy előtte ***CD-ROM***-ot is teszünk bele, azt kivehetjük, amint végeztünk, majd újraindíthatjuk a gépet. A bemutatott eljárás remekül működik régi laptopokkal is, feltéve ha van ***CD-ROM*** meghajtójuk és hálózati kártyájuk.

A kiszolgáló beállítása az ügyfelekhez

Az *Esteliben* levő rendszerünk most kilenc ügyfelet szolgál ki, némelyikük rendelkezik *boot-ROM PXE*-vel, a többi pedig különböző meghajtókkal imitál *boot-ROM*-ot. Ha *Flash* meghajtót használunk a célra, azt csak USB kapcsolaton csatlakoztatnunk kell a géphez indítás előtt. Mindegyik gép *BIOS*-át be kell állítanunk, hogy a rendszerindítás eszköze (boot device) a megfelelő meghajtó legyen. A *BIOS* elérésének módját lásd a gép indításakor a kijelzőn. A *„Press DEL to enter Set-up”* (*Nyomjon DEL-t a Setup-ba való belépéshez*), vagy valami hasonló sort kell keresni. A megfelelő billentyű, vagy billentyűkombináció megnyomásával elérhetővé válik a *BIOS* menürendszere. Az indítóeszközök sorrendjét többnyire a második menüpontban állíthatjuk be, amelynek a neve *„Advanced BIOS options” (Különleges BIOS beállítások)*, vagy valami hasonló. Ebben a menüben kell választanunk az első rendszerindító eszközt. *Flash* meghajtó esetén az *USBHDD* lehet a megfelelő választás.

Az ügyfélnek szereznie kell egy rendszermagot a kiszolgálótól. A rendszermag alkotja az operációs rendszert és teszi lehetővé más programok futtatását. Ahhoz, hogy az ügyfél gépek megkapják ezt a rendszermagot, azonosítaniuk kell magukat és egy hálózati címet kérniük. Minden ügyfél elküldi egyedi azonosítóját, ami valójában a hálózati kártya *MAC* címe, a kiszolgáló pedig kioszt nekik egy hálózati címet, azaz *IP (Internet Protokoll)* címet. Az ügyfelek a címet egy hálózati protokollon keresztül kapják meg a kiszolgálótól – ez a *Dynamic Host Configuration Protocol (DHCP)*. A *PXE*-vel induló ügyfelek automatikusan kapnak egy a *dhcpd.conf*-ban megadott dinamikus *IP* címet. A *PXE* nélküli ügyfelek állandó címet kapnak a dhcpd.conf egy beállítása alapján, hogy létrejöhet a hálózati kapcsolat és megkaphatják a *Linux* rendszermag képfájljukat. Azt tapasztaltuk, hogy bizonyos gépek csak *vmlinuz* nevű képfájlokkal működnek, *bzImage* nevűekkel nem.

Minden *PXE* nélküli ügyfél *MAC* címét, állandó *IP* címét és a számára kiosztandó rendszermag képfájl nevét beírtuk a kiszolgáló */etc/dhcpd.conf* fájljába. Néha a *Linux Terminal Server Project (LTSP)* csomagot is be kell állítanunk, amely a hálózatunk fájlrendszer felépítését biztosítja. Az *ltsp.conf* beállításfájlt kell módosítanunk, ha valamelyik ügyfél olyan egeret, billentyűzetet, vagy monitort használ, amelyet az *LTSP* nem ismer fel automatikusan. Itt látható a kiszolgálónk *dhcpd.conf* fájljának egy része:

```
ddns-update-style ad-hoc;
allow booting;
allow bootp;
subnet 198.186.207.0 netmask
➥255.255.255.0 {
range dynamic-bootp
➥198.186.207.205
➥198.186.207.220;
default-lease-time 21600;
max-lease-time 43200;
}
next-server 198.186.207.124;
filename "pxelinux.0";
option root-path
➥"198.186.207.124:/opt/ltsp/
➥i386";
host ws001 {
hardware ethernet    00:11:
➥5B:86:46:B5;
fixed-address        198.
➥186.207.201;
filename             "/lts/
➥vmlinuz-2.6.9-ltsp-3";
}
host ws002 {
hardware ethernet    00:60:
➥08:C6:2B:43;
fixed-address        198.186.
➥207.202;
filename             "/lts/
➥vmlinuz-2.6.9-ltsp-3";
}
```

Itt pedig a *ltsp.conf* fájl fő része:

```
[Default]
  SERVER             = 198.186.
                     ➥207.124
  XSERVER            = auto
  X_MOUSE_PROTOCOL   = "PS/2"
  X_MOUSE_DEVICE     = "/dev/
                     ➥psaux"
  X_MOUSE_RESOLUTION = 400
  X_MOUSE_BUTTONS    = 3
  XkbLayout          = es
  USE_XFS            = N
  SCREEN_01          = startx
```

A *LTSP* csomag lehetővé teszi az egyes ügyfeleken futó multimédia és egyéb alkalmazások igen finom beállítását. Ezek a beállítások az *ltsp.conf*-ban találhatók a *dhcpd.conf*-hoz hasonlóan. A mi beállításaink egyszerűek, mivel az ügyfelek főként böngésznek és irodai alkalmazásokat futtatnak. Az *LTSP* már az alapbeállításokkal felismerte az ügyfelek összes hardvereszközét, kivéve a billentyűzetkiosztást, de azt egyetlen sor megadásával spanyolra állíthattuk.

A bemutatott technológia könnyen elérhető és rendkívül rugalmas. Segítségével itt *Nicaraguában* régi eszközökkel, de a legújabb szoftverekkel taníthatunk számítógépes ismereteket nagyszámú rossz anyagi helyzetű tanulónak. A szabadon elérhető eszközöknek és a jó dokumentációnak köszönhetően még egy kezdő *Linux* felhasználó is létrehozhat egy hasonló, lemez nélküli ügyfél rendszert, oktatási, kereskedelmi, vagy adminisztrációs célokra. A minimális befektetéshez képest a megtérülés pedig igen jelentős.

Linux Journal 2006., 142. szám

Stephen Sefton

Ír állampolgár. Közel húsz évig dolgozott közösségfejlesztő munkásként Közép-Amerikában, többek között az emberi jogok, az egészségügy és a fenntartható mezőgazdaság területén. Ha hazalátogat Wexfordba, Christy bácsikájával megnézi az öblöt Rosslare Strandban, és eltűnődik azon, mennyi idő alatt csúszik a szikla a tengerbe. Visszaemlékezik a Nicaragua-tó déli partján és a Curraghcloe mentén tett hosszú sétákra.

KAPCSOLÓDÓ CÍMEK

➲ www.linuxjournal.com/article/8699

Démonok harca – Gyakorlati útmutató az OpenSSH beállításainak megerősítéséhez

Elegendő néhány egyszerű trükk, és nyugodtan hajthatjuk álomra a fejünket: nem kell félnünk attól, ki tör be az éjjel féltve őrzött SSH kiszolgálónkra.

Ha egy *Linux* gépet kell felügyelnünk, akár hivatalból, akár mert a miénk, jó esély van arra, hogy olykor-olykor távolról be kell jelentkeznünk, rosszabb esetben folyamatos kapcsolatra lehet szükség. Legyen szó otthoni munkaállomásról, munkahelyi kiszolgálóról vagy hobbigépről, majdnem biztos, hogy a *Linuxhoz* való távoli hozzáférés esetén a távoli gépen *OpenSSH* kiszolgáló, a helyi gépen pedig valamilyen *SSH* ügyfél fut. (Ha mégsem, akkor pedig különösen ajánlom a cikk elolvasását.) Igaz ugyan, hogy az *SSH* kiszolgálók és ügyfelek hihetetlen mennyiségű forgalom titkosításáról gondoskodnak, de az is tény, hogy egyben minden kiszolgáló ajtó is, melyet a rossz fiúk szeretnének kinyitni. Közülük a legrosszabbak azok a démonnak nevezett egyének vagy csoportok, akik rossz szándékkal keresik a nevükben azonos kiszolgáló programokat.
Ez utóbbi démonok, a beállításoktól függően, vagy biztonságosak, vagy nem. Míg a hackerek általában szakmai büszkeségétől vezérelve törnek be gépekre, a crackerek infohuligánok, akik ugyanezt aljas szándékkal követik el. Így vagy úgy, egyik sem kívánatos jelenség. A továbbiakban az *OpenSSH* alapbeállításából kiindulva bemutatom, hogyan kell azt módosítani a számítógépünk védőbástyáját jelentő *OpenSSH* démon biztonságossá tételéhez.
A mi gépünkön lévő biztonsági rés ráadásul mások számítógépeire is veszélyt jelenthet. Az Internet a bolygónkon található adattovábbító megoldások egyik legveszélyesebbike.
Az *OpenSSH* segítségével megoldható a biztonságos kommunikáció a nem biztonságos csatornán. A beállítások főszereplője az *sshd_config* fájl, amely számos kapcsolót tartalmaz a biztonság fokozásához. Azt gondolhatnánk, hogy egy távoli hozzáférést biztosító eszköz eleve biztonságos, de ez sajnos távolról sincs így. Az *OpenSSH* kiszolgáló telepítését követően egy olyan, viszonylag biztonságos beállításunk lesz, amit a rendszergazda azért még jelentősen javíthat.
A távoli hozzáférés megtervezésekor a következő fő szempontokat kell figyelembe venni:

1. Kinek engedélyezzük a szolgáltatást?
2. Hogyan biztosítjuk a hozzáférést?
3. Honnan lehet igénybe venni?

A következőkben az *OpenSSH* kiszolgálót használjuk távoli hozzáférésre, így két kérdés marad megválaszolatlanul: kinek engedélyezzük a hozzáférést és honnét? A válasz lehet nagyon egyszerű, például egyetlen felhasználónk van egy adott tartományban. Lehet azonban meglehetősen komplikált is, ha például több, gyakran utazó felhasználónk van.

Az ördögi parti résztvevőlistája

Közismert, hogy egy *Linux* gép első és legfontosabb felhasználója a root, és remélhetőleg az is közismert, hogy ha root jogosultságokra van szükségünk, annak számos jobb módja van, mint *SSH*-val szimplán rootként bejelentkezni. Elég a nyers erővel elkövetett betörési kísérletre gondolnunk, melynek a root a legnyilvánvalóbb célpontja. A felhasználói név adott, azon már nem is kell gondolkodni. Az *sshd_config* fájlban a `PermitRootLogin` direktívával megtilthatjuk a rootnak a bejelentkezést.
A távoli hozzáférésre használt felhasználóknál nagyon ül a régi mondás: ***„A legjobb gyógymód a megelőzés.”***
Az *sshd_config* fájl `UsersAllow` és `UsersDeny` kapcsolóinak alkalmazása már több mint megelőzés, és bár ez minden egyes felhasználó felvételénél plusz egy lépést jelent, a `UsersAllow` módosítása azt jelenti, hogy a gyógymód ismeretére valószínűleg és remélhetőleg sohasem lesz szükség.
A `UsersAllow` direktívába nem csak csupasz felhasználóneveket tehetünk, hanem gépnévhez kapcsoltakat is. Tehát ha előre tudjuk ki és honnan szeretne a gépünkre bejelentkezni, a nyugalmunk érdekében minimális időráfordítással eszközölhetjük ezeket a restrikciókat az *sshd_config* fájlban.
A létező felhasználók listája az */etc/passwd* fájlban, illetve *NIS* vagy *LDAP* környezetben az ezzel egyenértékű állományban található. Az *1. Lista* bemutatja, hogy az elmúlt hónap biztonsági naplófájljaiban hogyan kereshetjük meg azokat a felhasználókat, akik sikeresen jelentkeztek be *SSH*-val.

Tartsuk szemmel és távol a démonokat

A démonokat kereső démonok figyelése viszonylag egyszerű, és a kapcsolódó alapbeállításokat mindenképp ajánlott átvezetni az *sshd_config* fájlba. Ha a `SyslogFacility` értéke `AUTH`, akkor az *sshd* démon a rendszernaplózás segítségével bejegyzéseket készít

1. Lista Sikeres bejelentkezések keresése a naplókban

```
cat secure* | grep Accepted | awk -F' ' '\
{print $1" "$2" "$9}' | uniq -u
Aug 30 juser
Aug 22 kuser
Aug 23 user
Aug 15 foo
...
Aug 24 13:23:19 foohost sshd[16348]: Accepted
password for phil from 127.0.0.1 port 47338 ssh2
Aug 24 13:23:25 foohost sshd[16398]: User root
not allowed because not listed in AllowUsers
```

2. Lista Az sshd_config fájlban nem engedélyezett felhasználók belépési kísérleteinél keletkezett naplóbejegyzések

```
User mail not allowed because not listed in
AllowUsers
User adm not allowed because not listed in
AllowUsers
```

3. Lista Nem létező felhasználók belépési kísérletéből származó naplóbejegyzések

```
Aug 28 06:04:15 foo sshd[11602]: Failed password
for illegal user a... from 10.0.0.1 port 35078 ssh2
Aug 28 06:04:17 foo sshd[11604]: Illegal user aaa
from 10.0.0.1
Aug 28 06:04:19 foo sshd[11604]: Failed password
for illegal user aaa from 10.0.0.1 port 35417 ssh2
Aug 28 06:04:21 foo sshd[11606]: Illegal user qqq
from 10.0.0.1
```

a */var/log/messages* és a */var/log/secure* fájlokba (az útvonalak és a fájlok neve függhet a disztribúciótól). Melegen ajánlom, hogy a rendszernaplót a ***Psionic*** naplóböngészőjéhez hasonló eszközzel nézzük meg, ez ugyanis értelmezi a naplóállományt, így képesek leszünk felismerni az ***sshd*** által engedélyezett és a sikertelen bejelentkezéseket egyaránt. Nyilvánvaló különbség van az engedély nélküli és a sikertelen belépés között. Utóbbi azt jelenti, hogy a gépen létező felhasználói fiók tulajdonosa sikertelenül próbálkozott, és az előző pont ilyen lehet. Ez meglehetősen fontos lehet akkor, amikor felhasználóneveket választunk, és azon elmélkedünk, ki kerüljön az `AllowUser`, és ki a `DenyUser` listára. Például lehet, hogy nincs `amanda` felhasználónk, de használjuk az `amanda` nevű nyílt forráskódú archiválóprogramot. Ha ez a név nem kerül a `DenyUser` listára, akkor a sikertelen bejelentkezések számát fogja növelni, nem pedig a nem létező felhasználókét. Biztos, ami biztos, a rendszer felhasználói fiókjait mindig a `DenyUsers` listára teszem, még akkor is, ha ez nem kifejezetten szükséges. A *2.* és a *3. Listában* az ***sshd*** naplójának a legfrissebb bejegyzései arról árulkodnak, hogy a rendszer felhasználói fiókjaival próbálkoztak.

Ha már úgyis a felhasználóneveknél tartunk, érdemes megemlékezni annak a lehetőségéről, hogy a gonosztevő démonok a belépéshez szükséges két dolog egyikével rendelkeznek. Egy felhasználóknak helyet adó webkiszolgáló esetében vagy az elektronikus levelek fejlécéből viszonylag egyszerűen hozzájuthatunk egy bizonyos géphez tartozó felhasználónevekhez. A démon ezzel a felhasználónév-jelszó páros egyik felének máris a birtokában van, így a támadás túllépheti a jól ismert felhasználói fiókok feltörésén, és következhetnek a nyers erővel folytatott kísérletek azokkal a felhasználónevekkel, amelyekről kiderült, hogy az adott számítógépen valóban léteznek.
Nem kell újra felfedezni a protokollt, elegendő a beállítások finomítása. Az ***OpenSSH*** erős titkosítást biztosít a hálózatba kötött végpontok között. A fel-felbukkanó biztonsági réseket folyamatosan javítják és elérhetővé teszik. Az egyik legfontosabb kiadás az ***SSH v1***-ről az ***SSH v2***-re váltás volt. A v1-es és a v2-es sshd démonnak is saját kulcsa van, más szóval, az ***SSH1*** és az ***SSH2*** nyilvános és titkos kulcsai nem keverednek, egymástól teljesen függetlenek. Ahogy a kulcsok, úgy a protokollverziók is függetlenek egymástól, és a konfiguráció `Protocol` direktívájával engedélyezhető az egyik, a másik, vagy mindkettő. Az ***SSH1*** már leáldozóban van, de még még mindig sok szervezetnél és alkalmazásban használatos. Érdemes ezt ellenőrizni az ***sshd_config*** fájlban, és letiltani az v1 verziót, amennyiben nincs rá szükség. Így nyilvánvalóan kiküszöbölhető, hogy a v1-es változat egy biztonsági hibájának áldozatául essünk.
A ***passwd*** fájl korábbi átvizsgálásánál talán szemet szúrt az ***sshd*** felhasználó, */var/empty* saját könyvtárral. Ha nincs, akkor a következő sorok különleges fontosak lesznek, és az Olvasó bizonyára további részleteket is szeretne megtudni az ***sshd*** többfolyamatú, privilégium-szeparált módjáról. Az ilyen módban futó ***sshd*** démon létrehoz egy privilegizált monitorfolyamatot, amely egy újabb ***sshd*** folyamatot indít a felhasználó jogosultságaival, és végül ez utóbbi indítja a parancssori értelmezőt. A privilégiumszeparáció a ***chroot***-tal valósul meg és a */var/empty* könyvtárra korlátozódik. Pontosan

ugyanarról van szó, mint a jogosultságok szétválasztásánál a ***chroot*** használatakor. A hallgatózó démon számára védelmet biztosít memóriatúlcímzés vagy más hasonló támadási kísérlet esetén. A /var/empty könyvtár legyen üres, tulajdonosa legyen a root, és a csoport, illetve a többi felhasználó számára ne legyen írható.
A privilégiumszeparáció az ***sshd*** felhasználó nélkül nem működik, és azoknál a rendszereknél, amelyek nem támogatják az ***mmap***-et vagy az ***anonymous*** memória elosztását, a tömörítést nem szabad engedélyezni.
Tegyük fel, hogy csak néhány felhasználónk van, akik ***SSH***-val szeretnének a gépre bejelentkezni, és az adminisztratív feladatok ellátását a belépés után a `su` vagy `sudo` parancsokkal végezzük el. Vegyük most azokat a szkripteket, amelyekkel az ***SSH***-t futtató kiszolgálókat próbálják megtalálni. Lehet, hogy a rendszer összes portját végigpásztázzák, áldozatot keresve, de még valószínűbb, hogy csak egyetlen port, az ***sshd*** esetében a 22-es lesz a célpont, és ha nyitva van, itt folytatódik a támadás. A kevés felhasználót kiszolgáló ***sshd*** démonok esetében tehát célszerű megváltoztatni az alapértelmezett 22-es portot, alaposan megtréfálva ezzel a szkriptek jelentős részét. Tűnjön bármily egyszerűnek ez a kis változtatás, ez fogja a nyers erővel elkövetett támadásokat a legdrámaibb mértékben visszaszorítani (*4. Lista*).
Mennyi idő kellene a világ leglassabban gépelő emberének, hogy bepötyögje a jelszót? Nyolc karakteres jelszó esetén 8 másodperc? Adjunk neki karakterenként még egy másodpercet, és legyen összesen 16 másodperc. Számoljunk hozzá újabb 20 másodpercet a hálózati késleltetés miatt, ami már több mint fél perc – igazán nagyvonalú ajánlat egy jelszó begépelésére. Az ***sshd_config*** alapértelmezett `LoginGraceTime` direktívája gyakran ezt is felülmúlja, az akár 2 percet is elérő értékével. Kérdezzük meg magunktól, mennyi ideig gondolkodnánk, hogy beengedjünk-e valakit, miután az ajtón kopogtatott? Az esethez egy másik speciális direktíva is kapcsolódik, a `MaxStartups`, amely komoly fegyver lehet a ***SSH***-portpásztázókkal szemben. Ez korlátozza az egymást követő sikertelen bejelentkezések számát. Figyeljünk arra, hogy ez engedély nélküli kapcsolatokat is jelent. Ha egy szkript nyers erővel próbál a kiszolgálóra betörni, és képes a bejelentkezési folyamatok elindítására, akár több száz folyamat is futhat párhuzamosan. Ökölszabály szerint, a `MaxStartups` értéke legyen a felhasználók számának a harmada, de nem több mint 10.

4. Lista Az SSH alapértelmezett portjának megváltoztatása

```
# Az OpenSSH-ban található sshd_config fájl
# kapcsolóinak esetében az a stratégia, hogy a
# kapcsolók az alapértelmezett értékkel szerepeljenek,
# amikor csak lehetséges, ugyanakkor megjegyzésben.
# Az alapértelmezett érték megváltoztatásához távolítsuk
# el a megjegyzésjelet és adjunk új értéket.
Port 13
Protocol 2
```

A változtatások aktiválása

Valójában nem sok mindenre kell figyelni a fentebb bemutatott változtatásokhoz. Az alapértelmezett beállítások csak útmutatásul szolgálnak, a felelősség a rendszergazdát terheli, hogy ezeket biztonság fokozása érdekében megváltoztassa. Mielőtt nekifognánk, indítsunk el egy második ***sshd*** démont egy másik porton, hogy legyen hol visszatérnünk, ha kívülről magunkra zártuk az ajtót. Az is előfordulhat, hogy a megváltoztatott beállítások miatt az sshd nem tud újra indulni. Ekkor az

```
ssh -p <másik port>
```

paranccsal egy második ***sshd*** démon is elindítható, amely kihúz bennünket a bajból. Végül is távolról dolgozunk, és ha elveszítjük a kapcsolatot, már csak a helyi konzolon köszörülhetjük ki a csorbát. Bár triviálisnak tűnik, mégis érdemes megemlíteni, hogy vagy `SIGHUP` szignált kell generálnunk, vagy újra kell indítanunk az ***SSH***-démont a változtatások életbe léptetéséhez. Továbbfűzve a másodlagos port gondolatát, olyan szkriptet is végrehajthatunk az indulásnál, amely egy második ***sshd*** démont is elindít olyan beállítással, hogy csak egy megadott felhasználó léphessen be egy megadott számítógépről. Ezzel akkor is biztosíthatjuk magunknak a bejutást egy biztonságosnak ítélt távoli gépről, ha az elsődleges ***sshd*** démon felmondja a szolgálatot.

A hajsza vége

Heroikus történetünkben a ***Linux*** rendszergazdák állnak szemben a világgal. Az egyik oldalon állunk mi, számítógépünkhöz hozzáférést biztosítva, a másikon pedig a hálózati kapcsolattal rendelkezők világa. Sokan közülük szeretnének bejelentkezni a gépünkre. Az ***SSH***-démon tulajdonságait kiaknázó eszközök gyakran az alapbeállításokat használják.
A célpontot rendszerint a fa alsó ágain lógó gyümölcsök testesítik meg, és a rendszergazda feladata, hogy ezekről tudjon és eltávolítsa, mielőtt illetéktelen kezekbe kerülnek. A gyenge védelemmel ellátott számítógép nem csak önmagára jelent veszély, hanem a hálózatba kapcsolt gépek millióira, milliárdjaira is. Az ***OpenSSH*** kiváló eszköz a távoli hozzáférés biztosításához. Leginkább egy állítható villáskulcshoz hasonlítható. Ahogy a kulcsot is állítani kell a különféle felhasználási céloknak megfelelően, az ***OpenSSH*** alapbeállításainak hangolása is szükséges a távoli hozzáférés lehetőségeinek legteljesebb és legbiztonságosabb szolgáltatásához.

Linux Journal 2006., 143. szám

Phil Moses napjait Linux rendszerek kezelésével tölti a Scripps Oceanográfiai Intézet Fizikai Oceanográfiai Kutatási Részlegén, szabadidejében pedig a világ kevéssé látogatott tájain barangol. Philnek a philmoses@cox.net címre lehet levelet küldeni.

Barkácsoljunk OpenSSL-lel

Az OpenSSL függvénykönyvtárhoz tartozik egy parancssori eszköz, amivel gyakorlatilag mindent kipróbálhatunk, amit a könyvtár lehetővé tesz.

Az *OpenSSL* a *Secure Sockets Layer (SSL)* kriptográfiai protokoll rendkívül jól használható implementációja. Az *Apache a HTTPS*, az *OpenSSH* az *SSH* megvalósításához használja. Bár függvénykönyvtárnak készült, sokoldalú, platformfüggetlen eszközként is hasznosítható.
Kezdetben volt *Eric A. Young SSL* implementációja: az *ssleay*. Jól sikerült továbbfejlesztésével később megszületett az *OpenSSL*, hasonlóan ahhoz, ahogy az *NCSA HTTPd* az *Apache* webkiszolgálóvá változott.
Az *OpenSSL* ma tucatnyi titkosító algoritmust és protokollt támogat, kapcsolók százával.
E rövid történeti áttekintés után, térjünk rá az *OpenSSL* tulajdonságainak bemutatására, melyek közül számos alkalmas *SSL*-ügyfél és -kiszolgáló megvalósítására. Ezen felül a következőket mondhatjuk el róla:

- Rendelkezik az *Egyesült Államok* szövetségi kormányának *NIST FIPS 140-2 1.* szintű minősítésével
- Támogatja az *SSL* következő generációját, a *TLS*-t
- Képes *X.509*-es kulcsot és bizonyítványt előállítani
- Képes *X.509*-es bizonyítványok tanúsítására
- Támogatja az *S/MIME* titkosítást
- Képes a fájlok titkosítására és lenyomatuk (*hash*) elkészítésére
- Képes a *UNIX*-jelszavak lenyomatának elkészítésére
- 9 kereskedelmi forgalomban kapható titkosító hardvereszközt támogat
- Segítségével kriptográfiai teljesítménytesztek végezhetők
- 36 paranccsal vezérelhető
- 6 algoritmust tartalmaz üzenetpecsétek készítéséhez
- 9 titkosító algoritmust támogat, összesen 4 blokkmóddal
- Több kriptográfiai protokollt támogat

Bár az *OpenSSL* meglehetősen bonyolult, ennek nagy részével nem kell szembesülnünk. A cikk hátralévő része a könnyen használható tulajdonságokat mutatja be, mindössze néhány parancssorral.
Most is a korábbi, *GnuPG*-ről írt cikkemben használt fejezetcímeket használom, az *OpenSSL* és a *GnuPG* összehasonlításának megkönnyítése érdekében (a másik cikk *GnuPG trükkök* címmel jelent meg).

Az első lépések

Elsőként győződjünk meg róla, hogy az *OpenSSL* telepítve van és szerepel az elérési utak között. Sok *Linux* disztribúció, köztük néhány kisebb változat is alapértelmezésben tartalmazza az *OpenSSL*-t, amely a legtöbb kötelező csomaghoz hasonlóan a */usr/bin* könyvtárban található.
A parancssori értelmező készenléti jele a következő példák mindegyikében $.

Adjuk ki a következő parancsot:

```
$ openssl version
```

Figyeljünk rá, hogy a version kapcsoló előtt nincs kötőjel! Az eredmény valami ehhez hasonló lesz:

```
OpenSSL 0.9.7d 17 Mar 2004
```

A verziószám, a dátum és az egyéb részletek ettől eltérhetnek. A cikk írásakor a legfrissebb változat az *OpenSSL 0.9.8a* volt. A bemutatott példák minden bizonnyal működni fognak a legtöbb verzióban.
Ha kapcsolók nélkül adjuk ki az `openssl` parancsot, akkor a következőt látjuk:

```
OpenSSL>
```

Ez az *OpenSSL* beépített parancsértelmezője, melynek sem parancssori szerkesztője, sem közvetlen segítségnyújtása nincsen, de egy érvénytelen parancs kiadásakor kiírja az érvényeseket. Egyelőre hagyjuk békén.
Ha mégis ide jutottunk, a `quit` paranccsal vagy a *Ctrl+C* billentyűkombinációval léphetünk ki.

Bináris fájlok védelme

A bináris állományokat elektronikus levélben többnyire *MIME* protokollal küldjük. Ha azonban ezt a levelezőprogram nem támogatja, akkor marad a `uuencode` vagy az *OpenSSL base64* kódolása. Valójában a *MIME* is ez utóbbira épül, de sokkal bonyolultabb és nem is kompatibilis vele.
A következőképpen oldható meg egy fájl szöveges *base64* kódolása:

```
$ openssl base64 < filename.bin
➥> filename.txt
```

Ugyanez visszafelé:

```
$ openssl base64 -d <
➥filename.txt > filename.bin
```

Ne felejtsük el, hogy az *OpenSSL*-t cseppet sem érdekli a fájl kiterjesztése. Eltérően a *GnuPG*-től és a *MIME*-tól, az *OpenSSL* rövid szövegek kódolására is alkalmas:

```
$ echo "The Linux Journal" |
➥openssl base64
VGhlIExpbnV4IEpvdXJuYWwK
```

Ugyanez visszafelé, ahol a `-d` kapcsoló jelzi a dekódolást:

```
$ echo
➥"VGhlIExpbnV4IEpvdXJuYWwK" |
➥openssl base64 -d
The Linux Journal
```

Jobb ellenőrzőösszegek

A `sum` és a `cksum` hagyományos *UNIX* programok ellenőrzőösszegek kiszámítására. A célnak megfelelnek, amíg nincs szükség platformfüggetlenségre, biztonságra, és nem zavar bennünket, ha két különböző fájl esetén ugyanazt az értéket kapjuk.
Bár a legtöbb *Linux* rendszeren alapból megtalálható az `md5sum`, ezt egy nemrégiben felismert biztonsági rés miatt nem célszerű használni.
Ha a biztonságosabb sha1sum telepítve van, használjuk inkább azt. Vigyázzunk, mert számos különféle program fut ezen a néven. Némelyik egyszerre csak egy fájllal tud megbirkózni, esetleg nem tud adatot fogadni a szabványos bemenetről, vagy valami egészen más bajjal küszködik. Ha ilyen problémával szembesül, vagy egyszerűen csak egy konzisztens, közismert, platformfüggetlen szoftverre van szüksége, fontolja meg az *OpenSSL* használatát.
Az *OpenSSL* kimeneti formátuma kissé eltér *GnuPG*-étől, de tartalmilag természetesen megegyeznek. Az *OpenSSL* eredménye mindig tartalmazza a felhasznált algoritmust és a lenyomatot, csupa kisbetűvel, üres karakterek nélkül. Egyesek szerint ez a formátum könnyebben használható. Néhány példa:

```
$ openssl sha1 filename
SHA1(filename)=
➥e83a42b9bc8431a6645099be50b63
➥41a35d3dceb
$ openssl md5 filename
MD5(filename)=
➥26e9855f8ad6a5906fea121283c7
➥29c4
```

A hasonszőrű, *GnuPG*-ről szóló cikkem példáiban a fájl tartalma *„The Linux Journal”* volt. Fontos, hogy a *Journal* után nem szerepel pont.
Ha valami okból nem sikerül a fenti eredményeket reprodukálni, a fájl tartalmának hexadecimális leírása talán segíthet az ok megtalálásában. Érdemes odafigyelni a sor vége karakterre, amit a `vi` automatikusan adott a fájlhoz:

```
T h e   L i n u x
➥J o u r n a l \n
54 68 65 20 4c 69 6e 75 78 20
➥4a 6f 75 72 6e 61 6c 0a
```

Míg a *GnuPG*-ben van *SHA-512*, az *OpenSSL*-ben nincs, amit talán kárpótol a régi *MD2*, *MD4* és *MDC2* algoritmusok támogatása, a visszafele kompatibilitás érdekében. Az *MD5*-höz hasonlóan, már ezek használata sem ajánlott.

Gyors és egyszerű titkosítás

Az *OpenSSL*-lel fájlokat is titkosíthatunk, bár nem ez a legfőbb erénye. Rugalmassága miatt kissé bonyolultabb a használata, mint a *GnuPG*-é.
Nagyon kevés az alapértelmezett érték, így több kapcsolót kell használni. Több algoritmust is támogat, amik közül választani kell. Ezek közül néhányat, például a *DES*-t és az *RC4-40*-et csak a visszafele kompatibilitás érdekében tartottak meg, de használatuk többé nem ajánlott. Használjunk erős algoritmusokat, például a `bl`-t (*Blowfish*) és a 128 bites, a titkosító blokkok láncolásával működő `aes-128-cbc`-t (*US NIST Advanced Encryption Standard*).
Íme egy példa:

```
$ openssl enc -aes-128-cbc <
➥filename > filename.aes-128-
➥cbc
enter aes-128-cbc encryption
➥password:
Verifying - enter aes-128-cbc
➥encryption password:
```

A *GnuPG*-hez hasonlóan az *OpenSSL* is kétszer kéri a jelmondatot, amit nem jelenít meg a képernyőn.
A `-d` kapcsolóval kérhető visszafejtés is valamivel nehézkesebb:

```
$ openssl enc -d -aes-128-cbc
➥-in filename.aes-128-cbc >
➥filename
enter aes-128-cbc decryption
➥password:
```

A *GnuPG*-től eltérően, az *OpenSSL* nem találja ki magától a fájl típusát, az algoritmust, a kulcs hosszát és a módot. Ezt nekünk kell megtennünk. A fenti példában ezeket az adatokat a fájl kiterjesztésében helyeztem el. Az *OpenSSL* nem gondoskodik a fájlok és a kiterjesztések kezeléséről, nekünk kell megmondanunk, hová kerüljön az eredmény.
Ha az algoritmust nem adjuk meg, akkor az *OpenSSL* vagy szemetet ad, vagy hibás varázsszámra panaszkodik. A visszafejtés egyik esetben sem fog sikerülni. Őszintén szólva az *OpenSSL* nem igazán erre a célra készült, de erre is használható.

Jelmondatok

A lendület hevében időzzünk el egy kicsit a jelmondatok fontosságának témakörénél. A legtöbb kriptográfiai rendszerben létezik olyan speciális jelszó, az úgynevezett jelmondat (*passphrase*), amely további titkokat véd. Mivel ez a leggyengébb láncszem, fontos, hogy nehezen megfejthető legyen. Az erős jelmondat képzése azonban a megfelelő eszköz nélkül rendkívül nehéz feladat. Az *OpenSSL* egy ilyen megfelelő eszköz.
Az első ökölszabály, hogy az erős jelmondat hosszú, és a 8 karakter biztosan kevés (lásd az *1. Táblázatot*).
Az általános cél az, hogy olyan titok legyen a birtokunkban, amire könnyen emlékszünk, más nem tudja, nem képes kitalálni, és véletlenül sem ejti ki a száján.

Jelmondat képzése

Az *OpenSSL* segítségével gyorsan készíthetünk nagyon erős jelmondatokat, például:

1. táblázat *A jelmondatok erőssége, illetve a kitalálásukhoz szükséges idő nagysága (amely a körülményektől függően rendkívül változó lehet)*

Típus	Bájt	Karakter	Bit/karakter	Bithossz	Megfejtés ideje
Base64 [A-Za-z0-9+/=]	6	8	6	48	percek-órák
Base64 [A-Za-z0-9+/=]	9	12	6	72	évek
Base64 [A-Za-z0-9+/=]	12	16	6	96	évtizedek
Base64 [A-Za-z0-9+/=]	15	20	6	120	végtelen?
Diceware jelmondat		8 szó	12.9 / szó	120	végtelen?

```
$ openssl rand 15 -base64
wGcwstkb8Er0g6w1+Dm+
```

Minden futtatásnál más-más eredményt kapunk, hiszen a jelmondat véletlen alapú.
Az első paraméter (a példában 15) határozza meg, hány bájt hosszú legyen a generált jelszó, a második (a példában -base64) a karaktereket kódoló algoritmust állítja be. 15 byte esetén a kimenet mindig 20 bájt hosszú lesz, nem számítva a sor vége karaktert.
A base64 karakterkészletben megtalálhatók a kis és nagy betűk, a számok 1-9-ig, továbbá a +, a / és az = írásjelek. A karakterkészletet szándékosan korlátozták ezekre a jelekre, és a több nem is feltétlenül jobb. Biztonsági szempontból ez mindössze egyetlen karakter hozzáadásával kompenzálható, mert egy 8 karakteres ***ASCII*** jelszó körülbelül olyan erős, mint a kilenc karakteres base64-es társa.
Ha nem is olyan sebesen, mint az ***OpenSSL***, a ***Diceware*** erős és gyakran könnyen megjegyezhető jelmondatokat állít elő. Melegen ajánlom.

Titkosított jelszavak

Végre valami, amit a ***GnuPG*** már egyáltalán nem tud: az ***OpenSSL***-ben van egy beépített parancs, amivel pontosan olyan titkosított jelszavakat készíthetünk, mint amilyet a ***/bin/passwd***-vel.
Ha az Olvasót zavarja a szőrszálhasogatás, ezt a fejezetet inkább ugorja át.
Bár a ***Linux*** jelszavait gyakran titkosítottnak nevezik, azokról valójában lenyomat készül az ***MD5*** vagy a régi (***DES*** titkosító algoritmuson alapuló) ***UNIX*** jelszópecsét-algoritmussal. Ezáltal válik lehetővé, hogy a ***Linux*** akkor se tudja a jelszót, amikor megbizonyosodik arról, hogy a felhasználó helyesen adta meg. Amikor a felhasználó beállítja a jelszavát, arról lenyomat készül a ***/etc/shadow*** fájlba. Bejelentkezéskor a begépelt jelszóról ismét lenyomat készül, és ha ez megegyezik a ***/etc/shadow*** fájlban lévővel, akkor a kapu kitárul. Ellenkező esetben rossz jelszót adtunk meg, és a gép természetesen továbbra sem tudja a helyeset.
Nos, ezért rendkívül hasznos, ha saját jelszópecsétet készítünk. Tegyük fel, hogy egy másik számítógépen van szükségünk jelszóra. Lehet ez egy új felhasználói fiók miatt, vagy mert elfelejtettük és megkértük a rendszeradminisztrátort, hogy hozza alaphelyzetbe. Ha tudunk beszélni a rendszeradminisztrátorral, akkor könnyű helyzetben vagyunk, de mi van, ha mégsem? Lehet, hogy még sohasem találkoztunk vele korábban. Hogy beszéljük meg vele az új jelszót?
Az elektronikus levél nem biztonságos, de a telefon sem sokkal jobb.
A hagyományos posta napokig tartana, a biztonsági problémákról nem is beszélve. A fax, a szöveges üzenetküldő és a csipogó sem különb.
És még ennél is lehet rosszabb. Talán nem is bízunk meg a rendszergazdában. Való igaz, hogy ő a root felhasználó, de a jelszó ismerete túlmutat a hatáskörén. Lehet, hogy ugyanazt a jelszót akarjuk használni több számítógépen is, és azok rendszergazdáiban sem bízunk.
Szóval, ha paranoiásak vagyunk, tegyük a következőt:

```
$ openssl passwd -1
Password:
Verifying - Password:
$1$zmUy5lry$aG45DkcaJwM/GNlpBLT
➥DyO
```

A jelszót kétszer kell megadni, nem látszik a képernyőn. Ha több felhasználói fiókunk van, hajtsuk végre a parancsot többször. Az eredmény a jelszó véletlennel fűszerezett kriptográfiai lenyomata, így minden futtatáskor más és más eredményt ad, még akkor is, ha a jelszó ugyanaz. Példánkban a jelszó lenyomata:

```
$1$zmUy5lry$aG45DkcaJwM/
➥GNlpBLTDyO
```

Ha ugyanezt kipróbáljuk, a kezdeti `$1$` karaktereket leszámítva teljesen más kimenetet kapunk.
A lenyomatot nyugodtan faxolhatjuk, elküldhetjük elektronikus levélben, szöveges üzenetben, de akár szóban is megadhatjuk a rendszergazdának, hogy ezt állítsa be jelszópecsét gyanánt a ***/etc/passwd*** fájlban manuálisan vagy a `chpasswd` paranccsal. Utóbbi esetben szükség van egy ideiglenes fájlra, nevezzük `newpassword`-nek, benne a felhasználónévvel és a jelszólenyomattal, például:

```
felhasznalonev:$1$ywrU2ttf$yjm9
➥OXTIBnoKJLQK2Fw5c/
```

A fájlban több sor is szerepelhet, más felhasználói fiókokhoz.
A rendszergazda eztán root felhasználóként futtatja a következő parancsot:

```
chpasswd –encrypted <
➥newpassword
```

Az új jelszó beállítása ezzel befejeződött. Hacsak nem egy elképesztően erős jelmondatot választottunk, tanácsos a jelszót megváltoztatni az első belépéskor. Erre azért van szükség, mert a lenyomat ismeretében az eredeti jelszó a nyers erő stratégiájával megfejthető, de minél erősebb a jelszó, annál hosszabb idő alatt.
A jelszó megváltoztatásának ez a módja elég biztonságos. Ha például valaki megszerzi a jelszópecsétet, és megtudja, hogy melyik számítógépen található a hozzá tartozó felhasználói

2. táblázat *Lenyomatkészítés és blokkos titkosítás teljesítménye (a számok másodpercenként 1000 bájtban vannak megadva, 1024 bájtos blokkmérettel)*

	AMD K6-2 300MHz, Linux 2.6.12, OpenSSL 0.9.7g	AMD Athlon 1.333GHz, Linux 2.4.27, OpenSSL 0.9.7d	PowerMac G5 1.6GHz, Darwin Kernel Version 8.0.0, OpenSSL 0.9.7b
md5	26,733.93k	169,207.13k	76,921.71k
sha1	12,665.41k	113,973.25k	76,187.82k
blowfish cbc	9,663.40k	56,940.54k	44,433.14k
aes-128 cbc	5,134.78k	31,355.90k	54,987.78k

fiók, nem sokat ér vele, hiszen magát a jelszót csak a lenyomatot előállító személy tudja. A jelszópecsét akár egy népszerű magazinban is megjelenhet.
Apropó, a példa jelszava 28 véletlenszerűen választott base64 karakterből áll, így rendkívül nehezen törhető. Ennek ellenére ezt véletlenül se használjuk, mert a jelszó is szerepel a cikkben, itt:

```
HXzNnCTo8k44k8v7iz4ZkR/QWkM2
```

A jelszó és a lenyomat a következő parancsokkal készíthető el:

```
$ openssl rand 21 -base64
HXzNnCTo8k44k8v7iz4ZkR/QWkM2
$ openssl passwd -1
➥HXzNnCTo8k44k8v7iz4ZkR/
➥QWkM2
```

A példák a Linux rendszerekben használatos ***MD5*** lenyomatokat használják. Ha a régi ***UNIX*** jelszópecsétekre van szükség, egyszerűen hagyjuk el a -1-et:

```
$ openssl passwd
Password:
Verifying - Password:
xcx7DofWC0LpQ
```

A jelszó ebben a példában: ***TheLinux***

Kriptográfiai teljesítményteszt

Mivel az ***OpenSSL*** sok algoritmust támogat, jól használható kriptográfiai benchmarkok készítéséhez. A konzisztens kód alapján összevethető a kriptográfiai algoritmusok és hardverarchitektúrák teljesítménye. Ráadásul még beépített benchmark parancsa is van.

3. táblázat *Nyilvános kulcsú titkosítás teljesítménye*

	aláírás/s	ellenőrzés/s	
rsa 1024 bit	30.8	563.1	AMD K6-2 300MHz, Linux 2.6.12, OpenSSL 0.9.7g
dsa 1024 bit	61.6	50.7	AMD K6-2 300MHz, Linux 2.6.12, OpenSSL 0.9.7g
rsa 1024 bit	239.9	4,514.6	AMD Athlon 1.333 GHz, Linux 2.4.27, OpenSSL 0.9.7d
dsa 1024 bit	498.2	410.6	AMD Athlon 1.333 GHz, Linux 2.4.27, OpenSSL 0.9.7d

Az `openssl speed` parancs alapértelmezés szerint minden egyes algoritmust kipróbál minden létező móddal és kapcsolóval, különféle méretű bemeneti adatokkal. Ez utóbbi az algoritmusok induló vesztesége miatt fontos. A teljes benchmark teszt mintegy 6 percig tart, függetlenül a hardver teljesítményétől, és 124 sornyi teljesítményadatot állít elő, 29 sor összegzéssel.
Jegyezzük meg, hogy a kriptográfiai algoritmusok teljesítménye jelentősen függ az implementációtól. A nagyobb sebesség érdekében az ***OpenSSL*** kódja több algoritmusban is x86-os assembly részekkel tarkított. Más architektúrákhoz, például az ***ia64***-hez, a ***SPARC***-hoz és az ***x86-64***-hez sokkal kevesebb, míg megint másokhoz egyáltalán nem tartozik assembly kód. A gépi kódok a ***crypto/*/asm*** könyvtárakban találhatók. A 2. és *3. Táblázat* három különféle rendszer eredményeit mutatja.

Merre tovább

A cikk csak ízelítő abból, amire az ***OpenSSL*** képes a parancssorban. Található dokumentáció az ***OpenSSL*** weboldalán a dokumentumok, illetve a kapcsolódó anyagok címszó alatt, és több levelezőlista is a támogatás címszó alatt.
Az ***OpenSSL***-t ***C/C++***-ban írták ***C/C++***-hoz, de számos más nyelvre is adaptálták, így például ***Ruby***-ra. Mindezeken túl, a 2006 márciusában elnyert ***FIPS 140-2 1.*** szintű minősítés az ***OpenSSL***-t komoly versenyhelyzetbe hozta a kriptográfia vállalati és kormányzati piacán.

Linux Journal 2006., 147. szám

Anthony J. Stieber az informatikai biztonság szakértője és a UNIX rendszerek, a kriptográfia, a fizikai biztonság és néhány ránk nem tartozó dolog megszállottja. Mostanában az Egyesült Államokban, a Minnesota állambeli Minneapolisban melegszik. Ez a második cikke.

KAPCSOLÓDÓ CÍMEK

A cikkel kapcsolatos további anyagok a ➲ www.linuxjournal.com/article/9020 címen találhatók.

A hálózati környezet beállítása RTNETLINK-kel

Az RTNETLINK felhasználása a hálózati beállításokat befolyásoló alkalmazások fejlesztésére.

A *Linux* felhasználó terében futó alkalmazások a *NETLINK* segítségével kommunikálhatnak a rendszermaggal. A *NETLINK* a standard *socket* implementációjának kiterjesztése. Használatával üzeneteket válthatunk a rendszermag különféle komponenseivel, pl. a hálózatkezelő résszel, és befolyásolhatjuk is azok működését.
A cikkben azt fogom bemutatni, hogyan használható az *RTNETLINK*, a *NETLINK* hálózatkezelésért felelős része a hálózati környezet programozására. Áttekintjük az *RTNETLINK* leghasznosabb funkcióit, a releváns socketkezelő függvényeket, az *RTNETLINK* üzenetei összeállításának módját és végül néhány példakódot.
Az *RTNETLINK IPv4*-es részének neve *NETLINK_ROUTE*, az *IPv6*-os részé pedig *NETLINK_ROUTE6*.
A cikkben szereplő leírások mindkét verzióra érvényesek.
A hálózatiprotokoll-kezeléssel foglalkozó fejlesztők az *RTNETLINK* segítségével különféle hálózati komponenseket módosíthatnak és figyelhetnek meg, pl. az útválasztótáblákat és a hálózati interfészeket. Az internettechnológiák szabványosítását végző *IETF* (*Internet Engineering Task Force*) sok olyan specifikációt készített és készít, amely megvalósítható a felhasználói térben. Ezek többnyire megkövetelik az útválasztás módosítását és az értesülést a más folyamatok által eszközölt változtatásokról. Ezen protokollok egy részét a következő csoportokba sorolhatjuk:

- *Dinamikus útválasztó protokollok* – Ebbe a családba tartozik többek között a *Routing Information Protocol (RIP)*, az *Open Shortest Path First (OSPF)* és az *Exterior Gateway Protocol (EGP)*. Az útválasztáshoz szükséges információkat aktívan kezelik, miközben velük egyenrangú gépekkel és útválasztókkal kommunikálnak egy hálózatban vagy az Interneten.
- *Mobilitásprotokollok* – A mobil gépek különböző időpontokban különböző hálózatokhoz csatlakoznak, például a *Mobile IP (MIP)*, a *Session Initiation Protocol (SIP)* és a *Network Mobility (NEMO)* protokoll alkalmazásával, ami lehetővé teszi az útvonalválasztási információk folyamatos karbantartását és a kommunikáció folytonosságának biztosítását.
- *Ad hoc hálózati protokollok* – Ha nincs kiépített mobil hálózati infrastruktúra, vagyis pl. nincsenek útválasztók és *WLAN* hozzáférési pontok, a mobil eszközök közvetlenül, egyenrangú (*peer-to peer*) módon kapcsolódhatnak egymáshoz, miközben beállításaik eltérők. Egy földrengés súlytotta környéken vagy más vészhelyzetben praktikus lehet az ilyen protokollok, például az *Ad hoc On-demand Distance Vector (AODV)* vagy az *Optimized Link State Routing (OLSR)* használata, melyek megkövetelik az útválasztási információ karbantartását a más gépekkel való kommunikációhoz, miközben a szomszédos eszközöket útválasztóként vagy átjáróként használják.

Ezeknek a protokolloknak a felhasználói térben való megvalósítása segít a rendszermag kódjának túlzottan bonyolulttá válását elkerülni. Emellett egyszerűbb a fejlesztés és a tesztelés is, mivel számos fejlesztőeszköz áll rendelkezésünkre a felhasználói térben történő programozáshoz, és a rendszermag összeomlásától sem kell tartani a tesztelés vagy a végső felhasználás során.

Socketműveletek

A *socketek* lehetővé teszik két végpont kommunikációját, a programozói interfész standard adatstruktúra- és függvénykészletével. Az *RTNETLINK* esetében az egyik végpont a felhasználói térben, a másik a rendszermagban van. A hálózati környezet *RTNETLINK*-kel történő befolyásolásához a következő függvényhívássorozatra van szükség:

1. Socket megnyitása
2. A socket helyi címhez kötése (folyamatazonosító felhasználásával)
3. Üzenet küldése a másik végpontnak
4. Üzenet fogadása a másik végponttól
5. Socket bezárása

A `socket()` függvény megnyit egy végpontot, amely egyelőre sehová sem csatlakozik. A függvény prototípusa:

```
int socket(int domain, int
➥type, int protocol);
```

A `domain` adja meg a *socket* típusát, amely az *RTNETLINK* esetén *AF_NETLINK* (`PF_NETLINK`). A `type` a protokoll típusát határozza meg, lehet egyszerű (`SOCK_RAW`) vagy *datagram* (`SOCK_DGRAM`). A típus az *RTNETLINK-socket* szempontjából

irreleváns, bármelyiket megadhatjuk. A `protocol` határozza meg a socket ***NETLINK*** mivoltát, ami esetünkben a `NETLINK_ROUTE`. Ha sikerrel járt, a függvény pozitív egész számmal, a socketleíróval tér vissza, amit minden további ***RTNETLINK*** függvényhívásban használni fogunk, a socket lezárását is beleértve. Hiba esetén a visszatérési érték negatív, és az okot az ***errno.h*** fájlból elérhető `errno` változóból tudhatjuk meg.
A következő példa bemutatja, hogyan nyissunk meg egy ***RTNETLIK***-socketet:

```
int fd;
...
fd = socket(AF_NETLINK,
➥SOCK_RAW, NETLINK_ROUTE);
```

Ezt követően a socketet egy helyi címhez kell kötnünk. A felhasználói alkalmazások egyedi 32 bites azonosítót használhatnak erre a célra.
A függvény prototípusa:

```
int bind(int fd, struct
➥sockaddr *my_addr, socklen_t
➥addrlen);
```

A helyi címet a `sockaddr_nl` struktúrában kell megadni, amelynek deklarációja a ***linux/netlink.h*** fájlban található:

```
struct sockaddr_nl
{
  sa_family_t     nl_family; //
➥AF_NETLINK
  unsigned short  nl_pad;    //
➥zero
  __u32           nl_pid;   //
➥process pid
  __u32           nl_groups; //
➥multicast grps mask
};
```

Az `nl_pid` egyedi azonosító legyen, amire épp megfelelő a `getpid()` függvény eredménye, vagyis a socketet megnyitó folyamat azonosítója. Amennyiben több szál is fut, és mindegyik saját socketet nyit, módosított azonosítóra lehet szükség.
A struktúra kitöltését követően a kötés végrehajtható. Ha a `bind()` függvény hívása sikeres, a visszatérési érték nulla, ellenkező esetben negatív, és a hibakód a rendszerhibát tároló `errno` változóba kerül. Íme egy példa a `bind()` hívására:

```
struct sockaddr_nl la;
...
bzero(&la, sizeof(la));
la.nl_family = AF_NETLINK;
la.nl_pad = 0;
la.nl_pid = getpid();
la.nl_groups = 0;
rtn = bind(fd, (struct
➥sockaddr*) &la, sizeof(la));
```

Többescímzés esetén az `nl_groups` tagot is ki kell tölteni, úgy hogy a kívánt ***RTNETLINK*** művelet csoportjához tartozó csatlakozás megtörténjen.
Ha pl. értesülni szeretnénk arról, hogy más folyamatok piszkálták az útválasztótáblát, akkor az `RTMGRP_IPV4_ROUTE` és az `RTMGRP_NOTIFY` értékek vagy (|) kombinációját kell beállítanunk.
Az útválasztással kapcsolatos ***RTNETLINK*** üzeneteket a standard `sendmsg()` függvénnyel küldhetjük el a rendszermagnak. A függvény prototípusa:

```
ssize_t sendmsg(int fd, const
➥struct msghdr *msg, int flags);
```

Az `msg` mutató egy `msghdr` struktúrára mutat, melynek deklarációja így néz ki:

```
struct msghdr
{
  void *msg_name;        //
➥Címzett címe
  socklen_t msg_namelen; //
➥A címzett címének hossza
  struct iovec *msg_iov; //
➥Küldendő adatok tömbje
  size_t msg_iovlen;     //
➥Küldendő adatok száma
  void *msg_control;     //
➥Segédadatok
  size_t msg_controllen; //A
➥segédadatokat tároló buffer
➥hossza
  int msg_flags;         //
➥Kapcsolók a fogadott
➥üzenetekhez
};
```

Az `msg_name` egy `sockaddr_nl` struktúrára mutat, ami a `sendmsg()` függvény címzettjét tartalmazza. Mivel az üzenet a rendszermagnak szól, az `nl_family` tagot kivéve a `sockaddr_nl` struktúra minden mezője nulla lesz. Az `msg_namelen` tag a `sockaddr_nl` struktúra méretét kapja értékül.
Az `msg_iov` egy `iovec` struktúrára mutat, amelybe a művelet szempontjából releváns ***RTNETLINK*** üzenetet vagy üzeneteket töltjük. Az üzenetek számát az `msg_iovlen` tag határozza meg.
A többi mezőt nullára inicializáljuk.
RTNETLINK üzenetek vételére a `recv()` függvényt használjuk, melynek prototípusa:

```
ssize_t recv(int fd, void *buf,
➥size_t len, int flags);
```

A második változó egy `buffer` mutatója, ahová az érkező bájtok kerülnek, a harmadik ennek a buffernek a hosszát adja meg. ***RTNETLINK*** esetén a bájtok üzenetsorozatot rejtenek, melyet a ***netlink.h*** és az ***rtnetlink.h*** fájlokban definiált makrókkal fedhetünk fel. A `flags`-ben lévő kapcsolók a vétel természetét befolyásolják, de ***RTNETLINK***-nél bátran használhatjuk a nulla értéket.
A kommunikáció végén a socketet le kell zárni a `close()` függvénnyel, melynek prototípusa:

```
int close(int fd);
```

Az RTNETLINK funkciói

Az ***RTNETLINK***-et használó alkalmazások fejlesztőinek legalább a következő fejállományokat kell beemelniük:

```
#include <bits/sockaddr.h>
#include <asm/types.h>
#include <linux/rtnetlink.h>
#include <sys/socket.h>
```

Ezek a fájlok tartalmazzák a különféle adattípusok, struktúrák deklarációját, amik az ***RTNETLINK*** függvényhívásokhoz nélkülözhetetlenek. Következzék egy rövid leírás arról, hogy milyen, az ***RTNETLINK*** szempontjából releváns deklarációkat tartalmaznak ezek a fájlok:

- A ***bits/sockaddr.h*** fájlban találhatók a socket-függvényekben használatos címek deklarációi.
- Az ***asm/types.h*** fájl tartalmazza a ***NETLINK*** és az ***RTNETLINK*** állományaiban szereplő adattípusok deklarációját.

- A *linux/rtnetlink.h* fájl tartalmazza az *RTNETLINK*-ben használt makrókat és struktúrákat. Minthogy az *RTNETLINK* a *NETLINK*-re épül, a fájl beemeli a *linux/netlink.h* fejállományt. A *netlink.h*-ban találjuk meg a *NETLINK* általános makróit és struktúráit.
- A *sys/socket.h*-ban találhatók a socket megvalósításhoz kötődő függvényprototípusok és adatstruktúrák.

Az *RTNETLINK* használatával végrehajtható műveletek az *rtnetlink.h* állományban vannak felsorolva. Minden egyes művelet háromféle beavatkozást tesz lehetővé: hozzáadást, illetve frissítést (*NEW*), törlést (*DEL*) és lekérdezést (*GET*). A támogatott műveletek a következők:
A hálózati környezetet befolyásoló általános szolgáltatások:

- Adatkapcsolati szintű interfészbeállítások: `RTM_NEWLINK`, `RTM_DELLINK` és `RTM_GETLINK`
- Hálózati (*IP*) szintű interfészbeállítások: `RTM_NEWADDR`, `RTM_DELADDR` és `RTM_GETADDR`
- Hálózati (*IP*) szintű útvonalválasztási beállítások: `RTM_NEWROUTE`, `RTM_DELROUTE` és `RTM_GETROUTE`
- Az adatkapcsolati és hálózati címzés párosítására szolgáló gyorsítótár műveletei: `RTM_NEWNEIGH`, `RTM_DELNEIGH` és `RTM_GETNEIGH`

Forgalomszabályozó szolgáltatások:

- A hálózati réteg csomagjainak irányítása: `RTM_NEWRULE`, `RTM_DELRULE` és `RTM_GETRULE`
- A hálózati interfészek sorkezelésének beállítása: `RTM_NEWQDISC`, `RTM_DELQDISC` és `RTM_GETQDISC`
- A sorokban használt forgalomosztályok beállítása: `RTM_NEWTCLASS`, `RTM_DELTCLASS` és `RTM_GETTCLASS`
- A sorokban használt forgalomszűrők beállítása: `RTM_NEWTFILTER`, `RTM_DELTFILTER` és `RTM_GETTFILTER`

Az RTNETLINK üzeneteinek összeállítása és értelmezése

Az *RTNETLINK* kérés-válasz mechanizmussal cserél információt a hálózati környezetet befolyásolásához.
Az *RTNETLINK* kérése és válasza egyaránt üzenetstruktúrák sorozatából áll. Kérés esetén a struktúrát a hívó tölti fel, míg válasznál a rendszermag. Az *RTNETLINK* egy sor (`#define`) makrót kínál ezeknek a struktúráknak a feltöltésére, illetve a tartalmuk kinyerésére. Minden kérés az alábbi struktúrával kezdődik:

```
struct nlmsghdr
{
  __u32  nlmsg_len;   //Az
➥üzenet hossza, beleértve
➥a ezt a struktúrát is
  __u16  nlmsg_type;  //Az
➥üzenet típusa
  __u16  nlmsg_flags; //További
➥kapcsolók
  __u32  nlmsg_seq;   //
➥Sorozatszám
  __u32  nlmsg_pid;   //A küldő
➥folyamat azonosítója
}
```

Ez a *NETLINK* fejlécnek is nevezett struktúra határozza meg, hogy a kérés további részében milyen típusú *RTNETLINK* üzenetre számítsunk.
A mezők jelentése a következő:

- `nlmsg_len`: A teljes *RTNETLINK* üzenet hossza, beleértve az `nlmsghdr` struktúrát is. Kitöltését az `NLMSG_ALIGN(len)` makróval végezhetjük el, ahol len az `nlmsghdr` struktúrát követő üzenet hossza.
- `nlmsg_type`: 16 bites azonosító az üzenet típusának meghatározásához, például `RTM_NEWROUTE`.
- `nlmsg_flags`: 16 bites kapcsoló, amely tovább pontosítja az `nlmsg_type` mezőben meghatározott műveletet, például `NLM_F_REQUEST`.
- `nlmsg_seq` és `nlmsg_pid`: Ez a két mező egyértelműen azonosít egy *RTNETLINK* kérést. A hívó itt megadhat egy sorozatszámot és a folyamat azonosítóját.

Az `nlmsghdr` fejlécet a kérésben szereplő művelet szempontjából lényeges struktúrák követik. A művelet típusától függően, a hívónak az alábbi, *RTNETLINK* műveleti fejléceknek nevezett struktúrákból egyet vagy többet kell az üzenetben elhelyeznie:

- `rtmsg`: Ezt a struktúrát használjuk az útválasztótábla elemeinek megváltoztatására és lekérdezésére.
- `rtnexthop`: Az útválasztási bejegyzés következő csomópontja (next hop) a célállomás felé vezető úton elsőként szóba jövő gépet jelenti, amelyből egy bejegyzéshez több is tartozhat. Minden következő csomópontnak sokféle attribútuma lehet, például az *IP*-címe mellett a hálózati interfész is megadható.
- `rta_cacheinfo`: Minden útválasztási bejegyzéshez tartozik, többnyire a felhasználással kapcsolatos állapotinformáció, melyet a rendszermag rendszeresen frissít. Ezzel a struktúrával a felhasználó tájékoztatást kaphat az állapotinformációkról.
- `ifaddrmsg`: Ezzel a struktúrával módosíthatók vagy kérdezhetők le a hálózati interfészeknek a hálózati réteggel kapcsolatos attribútumai.
- `ifa_cacheinfo`: Az útválasztási bejegyzéshez hasonlóan, a hálózati interfészek is tárolnak magukról állapotinformációkat, amit a rendszermag frissít. Ezzel a struktúrával ezek az állapotinformációk kérdezhetők le.
- `ndmsg`: A struktúra a szomszédkeresés (*neighbor discovery*) nyomán létrejövő, a szomszédos gépek adatkapcsolati és hálózati rétegének címzése közötti kapcsolatok lekérdezésére és módosítására szolgál.
- `nda_cacheinfo`: A rendszermag által frissített szomszédkeresési bejegyzések adatainak tárolására szolgál.
- `ifinfomsg`: Ezzel a struktúrával a hálózati interfészek adatkapcsolati attribútumai kérdezhetők le és változtathatók meg.
- `tcmsg`: Ezt a struktúrát a forgalomszabályozás attribútumainak lekérdezésére és módosítására használjuk.

Az *RTNETLINK* művelet fejlécét a vonatkozó attribútumok követik, pl. az interfész száma és *IP*-címe, melyeket az `rtattr` struktúrában adunk meg. Minden attribútumhoz külön struktúra tartozik. Az `rtattr` típusa például a következő:

```
struct rtattr
{
  unsigned short  rta_len;
  unsigned short  rta_type;
};
```

Közvetlenül utána következik az attribútum értéke. Az *IPv4*-es cím pl. 4 bájt helyet foglal. Az `rta_len` ezt és az attribútumleíró struktúra hosszát is magában foglalja. Az `rta_type` az attribútum típusát határozza meg, értékeként az *rtnetlink.h* fájlban található felsoroló típusok tagjait veheti fel. Az `rtattr_type_t` és más felsoroló típusok adják meg azokat az attribútumazonosítókat, amelyek az `rta_type` mező értékei lehetnek, például `IFA_ADDRESS` és `NDA_DST`. Az egymás után fűzhető attribútumok számát az `RTATTR_MAX` makró korlátozza. Íme egy példa attribútum hozzáadására:

```
rtap->rta_type = RTA_DST;
rtap->rta_len = sizeof(struct
➥rtattr) + 4;
inet_pton(AF_INET, dsts,
     ((char *)rtap) +
➥sizeof(struct rtattr));
```

Az *RTNETLINK*-socketből származó információ szintén struktúrák sorozata. Legegyszerűbben úgy nyerhetjük ki az adatokat, hogy a bájtsorozat mentén egy mutatót mozgatunk, amit az éppen feldolgozott struktúra méretével mindannyiszor megnövelünk. Az eljárás egyszerűsítésére az *RTNETLINK* egy csokor makrót is nyújt:

- `NLMSG_NEXT(nlh, len)`: Eredménye a következő struktúrára mutat. `nlh` a legutoljára visszakapott fejléc mutatója, len pedig a teljes üzenet mérete. Ciklusban hívva, az összes üzenet kiolvasásárát lehetővé teszi.
- `NLMSG_DATA(nlh)`: A *NETLINK* fejlécben megadott műveletre vonatkozó *RTNETLINK* fejléc mutatóját eredményezi. Útválasztási bejegyzés manipulációja esetén a visszatérési érték egy `rtmsg` struktúra mutatója lesz.
- `RTM_RTA(r)`, `IFA_RTA(r)`, `NDA_RTA(r)`, `IFLA_RTA(r)` és `TCA_RTA(r)`: Eredményük az *RTNETLINK* üzenet `r` fejlécében megadott műveletre vonatkozó attribútumsor kezdetére mutat.
- `RTM_PAYLOAD(n)`, `IFA_PAYLOAD(n)`, `NDA_PAYLOAD(n)`, `IFLA_PAYLOAD(n)` és `TCA_PAYLOAD(n)`: Eredményük a *NETLINK* üzenet fejlécére mutató ***n*** által megadott *RTNETLINK* műveletet követő attribútumok teljes hossza.
- `RTA_NEXT(rta, attrlen)`: Az előzőleg megkapott attribútumot (`rta`) és a maradék attribútumok hosszát (`attrlen`) megadva a következő attribútum mutatóját adja eredményül.

Ha például egy útválasztótáblát kértünk le egy *RTNETLINK* kéréssel, a választ a következő módon dolgozzuk fel:

```
char *buf; // mutató az
➥RTNETLINK-adatra
int nll; // az összes adat
➥hossza bájtban
struct nlmsghdr *nlp;
struct rtmsg *rtp;
int rtl;
struct rtattr *rtap;
nlp = (struct nlmsghdr *) buf;
for(;NLMSG_OK(nlp, nll);
➥nlp=NLMSG_NEXT(nlp, nll))
{
  // az RTNETLINK-üzenet
➥fejlécének azonosítása
  rtp = (struct rtmsg *)
➥NLMSG_DATA(nlp);
  // az attribútumok kezdetének
➥megállapítása
  rtap = (struct rtattr *)
➥RTM_RTA(rtp);
  // az attribútumok hosszának
➥meghatározása
  rtl = RTM_PAYLOAD(nlp);
  // ciklus az összes
➥attribútum kiolvasásához
  for(;RTA_OK(rtap, rtl);
rtap=RTA_NEXT(rtap, rtl))
  {
    // attribútum feldolgozása
  }
}
```

Egy RTNETLINK példa sorról sorra

A most bemutatásra kerülő példakód az útválasztótáblán végrehajtható három műveletre koncentrál:

- `get_routing_table`: a rendszer fő útválasztótáblájának kiolvasása
- `set_routing_table`: új bejegyzés elhelyezése az útválasztótáblában
- `mon_routing_table`: a tábla változásainak megfigyelése

Mindhárom példa main() függvénye egy kaptafára készült, néhány további függvényt hív az *RTNETLINK* üzenetek létrehozásához, küldéséhez és a fogadott üzenetek feldolgozásához. Az egyszerűség kedvéért a hibakezelést teljesen mellőztem. A példák a *IPv4*-es környezetben működnek (`AF_INET`). Íme a main() függvény:

```
int main(int argc, char
➥*argv[])
{
  // socket megnyitása
  fd = socket(AF_NETLINK,
➥SOCK_RAW, NETLINK_ROUTE);
  // helyi cím beállítása
➥és kötése
  bzero(&la, sizeof(la));
  la.nl_family = AF_NETLINK;
  la.nl_pid = getpid();
  bind(fd, (struct sockaddr*)
➥&la, sizeof(la));
  // alfüggvények az RTNETLINK-
➥üzenetek létrehozására,
  // socketen küldésére, válasz
➥fogadására és feldolgozására
  form_request();
  send_request();
  recv_reply();
  read_reply();
  // socket lezárása
  close(fd);
}
```

Hasonlóképpen, a socketkommunikációt végző két függvény is majdnem teljesen azonos a példákban. Az egyik üzeneteket küld a rendszermagnak, a másik pedig üzeneteket fogad a rendszermagtól. A kivételt a `set_routing_table` és `mon_routing_table` példák jelentik. Az előbbiben nincs üzenetfogadási szakasz, míg az utóbbiban a küldés hiányzik, hiszen mindössze az útválasztási környezetben bekövetkező változások megfigyelése a feladat. A kérdéses adatokat a rendszermag többesküldéssel az összes olyan *RTNETLINK* socketnek eljuttatja, amely a megfelelő állapotban vár erre. Elsőként lássuk a kérelem küldését végző `send_request()` függvény kódját:

```
void send_request()
{
  // a távoli cím létrehozása
  bzero(&pa, sizeof(pa));
  pa.nl_family = AF_NETLINK;
  // a sendmsg() függvény
➥bemenetét képező msghdr
  // struktúra létrehozása és
➥inicializálása
  bzero(&msg, sizeof(msg));
  msg.msg_name = (void *) &pa;
  msg.msg_namelen = sizeof(pa);
  // az RTNETLINK-üzenet
➥mutatóját és méretét
  // az msghdr struktúrába
➥töltjük
  iov.iov_base = (void *)
➥&req.nl;
  iov.iov_len =
➥req.nl.nlmsg_len;
  msg.msg_iov = &iov;
  msg.msg_iovlen = 1;
  // az RTNETLINK-üzenet
➥küldése a rendszermagnak
  rtn = sendmsg(fd, &msg, 0);
}
```

Párja a fogadást végző recv_reply():

```
void recv_reply()
{
  char *p;
  // a socket kimeneti
➥bufferének inicializálása
  bzero(buf, sizeof(buf));
  p = buf;
  nll = 0;
  // addig olvasunk
➥a socketből, amíg NLMSG_DONE
  // típusú üzenetet nem
➥kapunk, vagy monitorozó
  // socketről van szó
  while(1) {
    rtn = recv(fd, p,
➥sizeof(buf) - nll, 0);
    nlp = (struct nlmsghdr *)
➥p;
    if(nlp->nlmsg_type ==
➥NLMSG_DONE)
      break;
    // a buffer mutatóját
➥a következő
    // üzenetre pozícionáljuk
    p += rtn;
    // a teljes hosszt
➥megnöveljük az utoljára
    // kapott üzenet méretével
    nll += rtn;
    if((la.nl_groups &
➥RTMGRP_IPV4_ROUTE)
                        ==
➥RTMGRP_IPV4_ROUTE)
      break;
  }
}
```

Mind az eddig bemutatott, mind a most következő függvények használnak globális változókat. Ezeket egyaránt használjuk a socketműveletekhez, illetve az ***RTNETLINK*** üzenetek elkészítéséhez és feldolgozásához:

```
// az RTNETLINK-kéréseket
➥tároló buffer
struct {
  struct nlmsghdr nl;
  struct rtmsg    rt;
  char            buf[8192];
} req;
// socketkommunikációhoz
➥használt változók
int fd;
struct sockaddr_nl la;
struct sockaddr_nl pa;
struct msghdr msg;
struct iovec iov;
int rtn;
// az RTNETLINK-válasz(oka)t
tároló buffer
char buf[8192];
// az RTNETLINK-üzenetek
➥feldolgozásánál
// használt üzenetmutatók
➥és -hosszak
struct nlmsghdr *nlp;
int nll;
struct rtmsg *rtp;
int rtl;
struct rtattr *rtap;
```

A get_routing_table példa az ***IPv4***-es hálózati környezet fő útválasztótábláját kérdezi le. A kérést összeállító form_request() függvény a következő:

```
void form_request()
{
  // a kérés bufferének
➥inicializálása
  bzero(&req, sizeof(req));
  // a NETLINK-fejléc
➥beállítása
  req.nl.nlmsg_len
              = NLMSG_LENGTH
➥(sizeof(struct rtmsg));
  req.nl.nlmsg_flags =
➥NLM_F_REQUEST | NLM_F_DUMP;
  req.nl.nlmsg_type =
➥RTM_GETROUTE;
  // az útválasztófejléc
➥beállítása
  req.rt.rtm_family = AF_INET;
  req.rt.rtm_table =
➥RT_TABLE_MAIN;
}
```

Az ***RTNETLINK*** kérésre érkező választ a buf változóban találjuk. Ezt a read_reply() függvénnyel feldolgozva megkapjuk az útválasztótábla tartalmát. A függvény kódja:

```
void read_reply()
{
  // az útválasztótábla
➥tartalmának (azaz egy
  // bejegyzésnek a tárolására
➥szolgáló karakterláncok
  char dsts[24], gws[24],
➥ifs[16], ms[24];
  // külső ciklus: bejárja az
➥összes NETLINK-fejlécet,
  // így az útválasztási
➥bejegyzését is
  nlp = (struct nlmsghdr *)
➥buf;
  for(;NLMSG_OK(nlp, nll);
➥nlp=NLMSG_NEXT(nlp, nll))
  {
    // az útválasztási
➥bejegyzés fejlécének
➥meghatározása
    rtp = (struct rtmsg *)
➥NLMSG_DATA(nlp);
    // csak a fő táblára
➥vagyunk kíváncsiak
    if(rtp->rtm_table !=
➥RT_TABLE_MAIN)
      continue;
    // a karakterláncok
➥inicializálása
    bzero(dsts, sizeof(dsts));
    bzero(gws, sizeof(gws));
    bzero(ifs, sizeof(ifs));
    bzero(ms, sizeof(ms));
    // belső ciklus: bejárja
➥egy bejegyzés összes
    // attribútumát
    rtap = (struct rtattr *)
➥RTM_RTA(rtp);
    rtl = RTM_PAYLOAD(nlp);
    for(;RTA_OK(rtap, rtl);
➥rtap=RTA_NEXT(rtap,rtl))
    {
      switch(rtap->rta_type)
      {
        // destination IPv4
```

```
➥address
        case RTA_DST:
          inet_ntop(AF_INET,
➥RTA_DATA(rtap),
                    dsts, 24);
          break;
        // next hop IPv4
➥address
        case RTA_GATEWAY:
          inet_ntop(AF_INET,
➥RTA_DATA(rtap),
                    gws, 24);
          break;
        // unique ID associated
➥with the network
        // interface
        case RTA_OIF:
          sprintf(ifs, "%d",
                  *((int *)
          RTA_DATA(rtap)));
        default:
          break;
      }
    }
    sprintf(ms, "%d", rtp->
➥rtm_dst_len);
    printf("dst %s/%s gw %s if
➥%s\n",
                    dsts, ms,
➥gws, ifs);
  }
}
```

A `set_routing_table` példában *RTNETLINK* kérést küldünk, hogy egy új bejegyzés kerüljön az útválasztótáblába. A kérdéses bejegyzés egy útvonal (32 bites hálózati előtaggal) egy privát *IP* címhez (192.168.0.100), a 2-es számú hálózati interfészen keresztül. Ezek az értékek sorra a `pn` (hálózati prefix hossza), `dsts` (cél *IP* cím) és `ifcn` (interfész száma) változókba kerülnek. Érdemes futtatni a `get_routing_table` példát, hogy képbe kerüljünk a rendszerünkben található interfészek azonosítószámát és az *IP* hálózatot illetően. A kérelem összeállítását végző `form_request()` kódja:

```
void form_request()
{
  // az útválasztási bejegyzés
➥attribútumai
  char dsts[24] =
➥"192.168.0.100";
  int ifcn = 2, pn = 32;
  // az RTNETLINK-kérés
➥bufferének inicializálása
  bzero(&req, sizeof(req));
  // a kérelem kezdeti
➥hosszának kiszámítása
  rtl = sizeof(struct rtmsg);
  // az első attribútum
➥hozzáadása:
  // a cél IP-cím beállítása
➥és az RTNETLINK-buffer
  // méretének növelése
  rtap = (struct rtattr *)
➥req.buf;
  rtap->rta_type = RTA_DST;
  rtap->rta_len = sizeof(struct
➥rtattr) + 4;
  inet_pton(AF_INET, dsts,
     ((char *)rtap) +
➥sizeof(struct rtattr));
  rtl += rtap->rta_len;
  // második attribútum
➥hozzáadása:
  // az interfész számának
➥beállítása és
  // a méret növelése
  rtap = (struct rtattr *)
➥(((char *)rtap)
            + rtap->rta_len);
  rtap->rta_type = RTA_OIF;
  rtap->rta_len = sizeof(struct
➥rtattr) + 4;
  memcpy(((char *)rtap) +
➥sizeof(struct rtattr),
          &ifcn, 4);
  rtl += rtap->rta_len;
  // a NETLINK-fejléc
➥összeállítása
  req.nl.nlmsg_len =
➥NLMSG_LENGTH(rtl);
  req.nl.nlmsg_flags =
➥NLM_F_REQUEST | NLM_F_CREATE;
  req.nl.nlmsg_type =
➥RTM_NEWROUTE;
  // a struct rtmsg fejléc
➥beállítása
  req.rt.rtm_family = AF_INET;
  req.rt.rtm_table =
➥RT_TABLE_MAIN;
  req.rt.rtm_protocol =
➥RTPROT_STATIC;
  req.rt.rtm_scope =
➥RT_SCOPE_UNIVERSE;
  req.rt.rtm_type =
➥RTN_UNICAST;
  // a hálózati prefix
➥nagyságának beállítása
  req.rt.rtm_dst_len = pn;
}
```

A `mon_routing_table` példa azokat az *RTNETLINK* üzeneteket fogadja, amik akkor keletkeznek, ha más folyamatok megváltoztatják a rendszer fő útválasztótábláját. Most is a korábban bemutatott `read_reply()` függvényt használjuk az üzenetek feldolgozására. A main() függvényben apró változtatást kell eszközölnünk. Mivel ez a művelet a rendszermag többesküldéssel továbbított üzeneteire figyel, a socket helyi címhez kötésénél az `RTMGRP_IPV4_ROUTE` és `RTMGRP_NOTIFY` kapcsolókat is használnunk kell:

```
la.nl_groups =
➥RTMGRP_IPV4_ROUTE |
➥RTMGRP_NOTIFY;
```

A `mon_routing_table` végrehajtása után adjuk ki a `route add` vagy `route del` parancsot egy másik parancssori értelmezőből, hogy lássuk az eredményt.

Összefoglalás

Az *RTNETLINK* egyszerű, ugyanakkor sokoldalú eszköz a *Linux* hálózati beállításainak manipulálásához. A felhasználó térben írt protokollimplementációk ideális alanyai az *RTNETLINK* használatának. Az *IPROUTE2*-nek is becézett, fejlett *IP* útválasztási parancsgyűjtemény is *NETLINK* alapú. Az *RTNETLINK* műveleteiről és kapcsolóiról többet is megtudhatunk a *NETLINK(7)* és az *RTNETLINK(7)* kézikönyvlapokból. A példakódok letölthetők az ➲ *ftp.ssc.com/pub/lj/listings/issue145/8498.tgz* címről.

Köszönetnyilvánítás

Hálásan köszönöm *Carmelita Goerg* professzor segítségét.

Asanga Udugama
(adu@comnets.uni-bremen.de) kutató, szoftverfejlesztő a Brémai Egyetem ComNets részlegén, Németországban. Jelenleg a mobilitással kapcsolatos, hálózati rétegbeli IETF-protokollok szabványosításában és implementációjában vesz részt. A nevét öregbíti néhány referenciaimplementációja is. Most épp álmai tolószékét várja (Meyra X3 szervóval).

D-Link
Building Networks for People

Hálózati eszközök (4. rész)
Vezeték nélküli hálózatok biztonsága

Tovább folytatjuk az otthoni hálózatunk építését: az előttünk álló legsürgetőbb feladat a vezeték nélküli adatkapcsolatunk biztonsági problémáinak megoldása.

A vezeték nélküli hálózatok meglehetősen védtelenek, fokozottan ki vannak téve a támadás veszélyének. Ennek oka az adatok továbbításában keresendő: mivel az információ vezeték helyett az éterben közlekedik, nem szükséges a 'fizikai' hozzáférés a hálózathoz. Elegendő csupán a rádiójelek terjedési hatósugarában lennünk, és elfoghatjuk a hálózat egyes gépei között gazdát cserélő információt. A dolog veszélyességét fokozza, hogy mi felhasználók egyébként is elég félvállról (gyakorlatilag semmibe) vesszük a biztonsági előírásokat, javaslatokat. Úgy gondoljuk, az emlegetett kockázat egyáltalán nem nyugszik valós alapokon – vagy szimplán csak nem érdekel bennünket. Ez a bizonyos kockázat a bekövetkezési valószínűség és az okozott kár szorzata. Azt tudjuk, hogy az adatforgalomhoz történő hozzáférés miatt a betörés bekövetkezési valószínűség jelentősen megnő, növelve ezáltal a kockázatot. Ez azonban még nem minden: idegen felhasználók hálózatunk használatához történő vonzalma abból ered, hogy mindennapi céljukra tudják használni azt. Ebből következően az okozott kár is jelentős lehet.

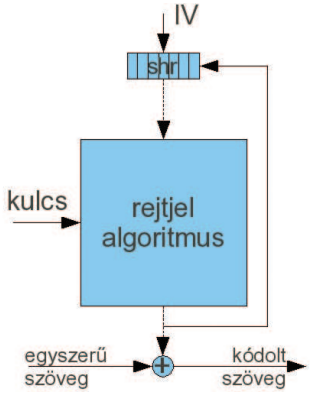
Hálózatunk illetéktelen használata a támadó fél számára az alábbi előnyökkel kecsegtet:

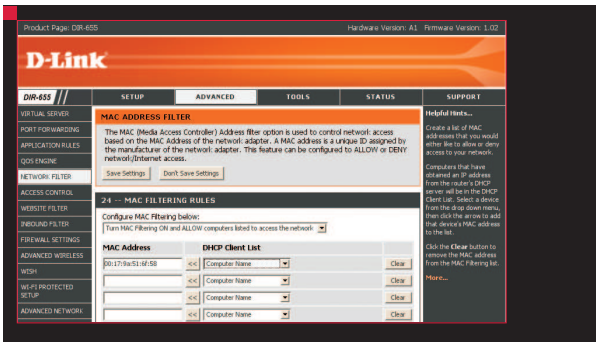
- Sikeres belépés után a mi előfizetésünket használva tud mindenféle illegális tevékenységet végezni: kéretlen reklámlevelet küldhet, másokat zaklathat, betörhet különböző kiszolgálókra, így a betörés nyomai hozzánk vezetnek, a betörő névtelen maradhat.
- A belső, otthoni hálózatunkon gyakran mindenféle korlátozás nélkül tesszük lehetővé a gépeken található dokumentumokhoz történő hozzáférést, ezáltal a hálózatunkba jutott ügyfélgépek egyszerűen lemásolhatnak fájlokat anélkül, hogy észrevennénk.
- Jelszavaink eltulajdonítása csak annyi munkába kerül, hogy el kell indítani egy programot, amely rögzíti a hálózaton küldött adatokat. Innen már gyerekjáték kiolvasni azokat. Leszámítva persze a titkos csatornákon történő adatátvitelt, de ez nem túl gyakori a mindennapokban.
- Sokan egyszerűen csak használják mások internet előfizetését a társasház szomszéd lakásából. Azon túl, hogy igazságtalan, és persze hogy benn van a hálózatunkban, letöltéseivel elhasználhatja a szolgáltató által rendelkezésre bocsátott letöltési kvótát, mi pedig hoppon maradunk.

Mondanom sem kell, hogy az utóbbi két eset igen kecsegtető és a bekövetkezése igen valószínű a mindennapokban is. Ennek megoldására két dolgot tehetünk. Az egyik, hogy lezárjuk a hálózatot akár jelszóval, akár egyéb technikával. Természetesen a forgalom figyelésével ez a védelem pillanatok alatt feltörhető, ezért el kell érni, hogy a forgalom a támadó számára értelmezhetetlen legyen, azaz kódolnunk kell az adatforgalmat. Az adatforgalom kódolása gyakran együtt jár a hálózat lezárásával, ugyanis a hálózatba történő belépéshez is kódolt adatokat kell kapnunk a hozzáférési ponttól.

A bukott szamár: WEP

A fentiekre természetesen a ***802.11***-es szabványcsalád kidolgozói is gondoltak, megalkották a ***WEP*** (***Wired Equivalent Protection;*** a vezetékessel egyenrangú védelem) titkosítást, ám a folyamat során valahol szőr került a hurkába, a protokoll ugyanis több durva hiányosságot is tartalmaz: nyíltan továbbítja a kulcs bizonyos részét (az úgynevezett inicializációs vektort), amelyek ráadásul 50%-os valószínűséggel már 5000 adatcsomag után ismétlődnek, ezen kívül az eredeti





A *WPA* és a *WPA2* kétféle autentikációs módot támogat: az egyik az otthoni használatra szánt ***PSK*** (***Pre-Shared Key***, előre kiosztott kulcs), a másik a főleg vállalati környezetbe illő ***Radius*** hitelesítő kiszolgálón keresztüli kulcskiosztó módszer, amely lehetővé teszi különböző hozzáférési szintek meghatározását is. Mi a beállításainkhoz a ***WPA-PSK*** protokollt fogjuk használni. Ennek lényege, hogy az autentikációt az induló kapcsolatkulcs, mint jelszó megadásával végezzük.

szabványban mindössze 40+24 bit volt a kulcs mérete. Ennek eredményeképp egy ilyen titkosítás feltörése az adatforgalom mennyiségétől függően kettő és tíz perc közé tehető. A probléma gyors orvoslása a kapcsolatkulcs méretének növelése 256bit-re, de ez csupán arra elég, hogy nem 10 perc, hanem 2-3 nap alatt tudja a támadó feltörni a hálózatot. *WEP* használata esetén a kapcsolatkulcs megadásával csatlakozhatunk a hozzáférési ponthoz.

Hatékony megoldás: WPA

A fenti hibák gyors megoldásért kiáltottak, megszületett a *WPA* (***Wi-Fi Protected Acces***, ***Wi-Fi*** védett hozzáférés), amely a *WEP*-hez hasonlóan ***RC4*** folyamkódolót használ 128 bites kulccsal és 48 bites inicializációs vektorral, ám lényeges különbség a ***TKIP*** (***Temporal Key Integrity Protocol***, ideiglenes biztonságos kulcs protokoll) bevezetése, amely folyamatosan cserélgeti a kapcsolat során használt kulcsot, így a támadó hiába fejtené meg a kulcsot, rövid időn belül semmire sem megy vele, pláne, ha egy kulcsot rövidebb ideig használ a rendszer, mint a feltöréséhez szükséges idő. A *WPA* előnye, hogy kompatibilis az összes korábbi eszközzel, hátránya, hogy ugyanazt az ***RC4*** folyamkódolót használja, mint a gyenge *WEP* protokoll, így még mindig nem nyújt tökéletes védelmet.

Biztos megoldás: WPA2

A ***WPA2*** gyakorlatilag egy időben készült a *WPA*-val, ezért is van, hogy a legtöbb implementáció egyesítve tartalmazza a ***WPA/WPA2*** protokollok kezelését. Legfőbb különbség a *WPA*-hoz képest az új ***AES*** (***Advanced Encryption Standard***, fejlett kódolási szabvány) kódoló használata a régi ***RC4*** helyett. Ezen kívül bevezették a négy lépéses azonosítási protokollt, ami nagyobb biztonságot nyújt a kapcsolódáskor történő támadások ellen. A ***WPA2*** „hátránya", hogy nem kompatibilis a régebbi (***820.11a,b***) eszközökkel illetve számos olcsóbb ***no-name g*** eszköz sem támogatja. A ***D-Link*** ilyen szempontból is jó választás, hiszen már a legolcsóbb, 10000 Ft alatti eszközök is tartalmazzák ezt a biztonsági módot.

Otthoni vezeték nélküli hálózatunk biztonságos beállítása

Otthoni hálózatunk feje még mindig a ***D-Link DIR-655***-ös szélessávú útválasztója, amelyet a ***D-Link Magyarországtól*** kaptunk a hálózatunk megépítéséhez. Ehhez kapcsolódunk a hasonló körülmények között birtokunkba került ***D-Link DWA-645***-ös *PC*-kártyával. Az útválasztó a fent ismertetett összes titkosítási módszert ismeri a visszamenőleges kompatibilitás miatt. A sorozat előző részében beállítottuk a második legerősebb biztonsági szintet, amely a *WPA* titkosítást alkalmazza. Azért ezt választottuk, hogy a közben eltelt időben is biztonságosan használjuk a hálózatot (említettük a *WEP* gyengeségeit), ugyanakkor a régi ügyfél oldali eszközökkel is hozzáférjünk.
Attól függően, hogy milyen ügyfél oldali eszközzel rendelkezünk, igyekezzünk az általa is támogatott legmagasabb biztonsági szintű kapcsolatot használni. Jelen esetben szerencsénk van, mivel a *PC*-kártyánk is a legújabb

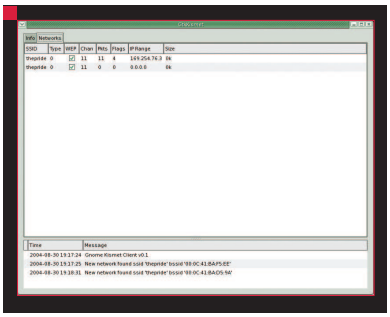
WPS: Biztonsági beállítások egyszerűen

A beállítási nehézségek kiküszöbölésére a ***Wi-Fi Alliance*** kitalált egy ajánlást, amelyet ***Wi-Fi Protected Setup***-nak (***Wi-Fi Védett Beállítások***) keresztelt. Az ajánlás lényege egy olyan egyszerű, egyetlen kattintással elvégezhető hálózatbeállítási, kapcsolatfelépítési folyamat, amely végén a felhasználó biztonságos hálózati kapcsolattal rendelkezik: nem kell jelszót és kapcsolatfelépítési információkat megjegyezni, a csatlakozáshoz elég megadnunk egy ***PIN*** kódot, és készen is vagyunk. A ***D-Link DIR-655*** útválasztó támogatja ezt az beállítási módszert, bár erről sem felhasználói kézikönyvben, sem a ***Wi-Fi*** kártya leírásában nem esik szó. Ez valószínűleg azért van, mert a vezeték nélküli hálózati kártyák vezérlőprogramjainak is támogatnia kell majd ezt a funkciót, s mivel a szabvány 2007 elején jelent meg, a hardvergyártóknak még nem volt idejük beépíteni. Mindenesetre a routerben a ***Wi-Fi Protected Setup*** menüpontra kattintva elvégezhetjük a beállításokat a hozzáférési pont oldaláról, a többi már a hálózati kártyák gyártóin múlik.

szériából való, így próbálkozhatunk a legerősebb, *WPA2* titkosítással. Ehhez nyissuk meg az eszközünk webes felületét, írjuk be a böngészőbe az *IP* címét: http://192.168.0.1 A bejelentkezés után válasszuk a bal oldali menüből a *WIRELESS SETTING* menüpontot, majd kattintsunk a *Wireless Network Setup Wizard* gombra, kövessük a varázsló utasításait, mindent változatlanul hagyva, és amikor a biztonsági szint képernyőhöz érünk, válasszuk a *BEST*, vagyis a legjobb lehetőséget, ezt követően adjuk meg a kapcsolódáshoz szükséges jelszót, és végül mentsük a beállításokat, és a megjelenő képernyőn válasszuk az eszköz újraindítását.
Ezek után máris csatlakozhatunk a *NetworkManager Applet* ikonjára kattintva. Az átállítás után az *Applet* még a régi titkosítási adatokat tárolja, ezért válasszuk a *Csatlakozás másik vezeték nélküli hálózathoz* menüpontot az elérhető vezeték nélküli eszközök listája alatt, adjuk meg a hálózatunk nevét, majd válasszuk a *WPA2* biztonsági szintet, adjuk meg a jelszót, ami egyben az induló kapcsolatkulcs (*Pre-Shared Key*). A típust hagyhatjuk *Automatikus* módban.

MAC korlátozás

Tovább fokozhatjuk a biztonságot, ha beállítjuk az útválasztón, hogy csak a meghatározott fizikai azonosítóval rendelkező hálózati eszközöket engedje csatlakozni. Minden hálózati eszköznek van egy egyedi FF:FF:FF:FF:FF:FF hexadecimális formátumú *MAC* címe, amelyet általában az eszköz hátoldaláról tudunk leolvasni, illetve az `ifconfig` parancs kiadásával kérhetünk le a számítógépen. Ehhez az útválasztó webfelületének *ADVANCED* lapján válasszuk a *NETWORK FILTER* menüpontot. A megjelenő képernyőn a legördülőmenüben válasszuk a második menüpontot, amely csak az itt listázott azonosítójú eszközöket engedi működni. Írjuk be az alább található beviteli mezőkbe a hálózatunkban használt vezetékes és vezeték nélküli eszközök *MAC* címeit, és mentsük a beállításokat.
A *MAC* cím persze egyszerűen hamisítható, ezért nem tökéletes védelem, de mégis eggyel több információ, amit a támadónak ismernie kell, ennélfogva növeli a hálózatunk biztonságát. Általában a *WPA* titkosítás is elegendő védelmet nyújt a támadások ellen, ezért optimális megoldás, ha így használjuk, ugyanakkor megengedjük az újabb eszközöknek, hogy használják az erősebb protokollt. Ehhez az útválasztó vezeték nélküli beállításainak lapján kattintsunk a *Manual Wireless Network Setup* (kézi beállítás) gombra, s a megjelenő képernyő *WPA Mode* feliratú lenyíló menüjében válasszuk az *Auto* módot, amely megengedi mind a *WPA*, mind a *WPA2* protokoll használatát.



Fix eszközeink beállítása WPA2 titkosításhoz

Az előző részben megnéztük, hogyan tudjuk a *NetworkManager Applet* hiányában, a gép indulásakor felélesztení a vezeték nélküli hálózatot az akkori beállításoknak, azaz a *WPA*-nak megfelelően. Ahhoz, hogy ez is *WPA2* protokollon kapcsolódjon, át kell írni a */etc/wpa_supplicant/wpa_supplicant.conf* fájl tartalmát az alábbira:

```
ctrl_interface=/var/run/
➥wpa_supplicant

network={
        ssid="a_hozzaferesi
➥_pont_neve"
        key_mgmt=WPA-PSK
        proto=WPA2
        pairwise=CCMP
        group=CCMP

psk="titkos_jelszo_ami_maga_a
➥_kapcsolatkulcs"
}
```

Az itt leírt megoldások persze nem bombabiztosak, idővel minden rendszer feltörhető, a kulcs itt az idő. Rohanó világunkban csak a gyors információ és a gyors cselekvési lehetőség ér valamit, ha csak lassan lehet megszerezni, az már nem is információ... A *WPA2* titkosítás *MAC* szűréssel ötvözve, rendszeresen változtatott jelszó (kulcs) használata esetén több, mint szükséges a számunkra, a hálózatunk biztonságosnak mondható.
A cikksorozat következő részében tovább csiszolgatjuk az otthoni hálózatunkat, kihasználva a DIR-655 útválasztó finombeállítási lehetőségeit: megnézzük, hogyan biztosíthatunk egyes gépeknek garantált válaszidőt, illetve beállítunk egy-két hozzáférési korlátozást, amivel például száműzhetjük az erőforrás-igényes és idegesítő reklámokat az otthoni hálózatunkból.

Feltöréshez szükséges szoftverek

A gyenge titkosítás feltöréséhez elegendő néhány ingyenesen elérhető szoftver, ami figyeli a hálózati forgalmat, és ezek alapján megszerzi a kapcsolatkulcsot. A szükséges szoftverek együtt, előre feltelepítve megtalálhatók az *Security Live CD* korongon, amelyet bárki letölthet a ➲ *http://www.knoppix.net/wiki/Security_Live_CD* címről. A dolgunk csupán annyi, hogy betöltjük a rendszert a *CD*-ről a gép indításakor, aztán használjuk az alábbi szoftvereket:

- *kismet*: azonosítja vezeték nélküli hálózatok adatait és résztvevőit rögzíti az adatforgalmat, segítségével összegyűjthető a támadáshoz szükséges adatmennyiség. Az adatforgalom rögzítéséhez az *Airodump* nevű szoftver is használható.
- *void11*: kijelentkezteti a hozzáférési pontra csatlakozott gépeket, így azok újracsatlakoznak, mi meg eltesszük ezeket az *ARP* kéréseket
- *arieplay*: a megszerzett *ARP* kéréseket tudjuk vele visszajátszani a hozzáférési pontnak, ezekkel növelve az adatforgalmat, és vele az inicializációs vektorok (*IV*) számát
- *aircrack*: az összegyűjtött forgalmi adatokból (az *IV*-k segítségével) megfejti a kapcsolatkulcsot

Biztonságosabb SSH kéttényezős hitelesítéssel

A következőkben ismertetem, hogyan kell létrehozni egy USB-minimeghajtó és az ssh-agent segítségével kéttényezős hitelesítést a rendszergazda bejelentkezéséhez.

Nagy rajongója vagyok a kéttényezős hitelesítésnek, és amikor csak lehet, azt használom, mert a statikus jelszavak nem épp a legbiztonságosabbak. A hagyományos jelszavak általában könnyen megfejthetők az emberek természetes, bizalomra való hajlamának kihasználásával, a kijelzők szélére ragasztott sárga cetlik tanulmányozásával, kulcsnaplózással és az egyre gyorsabb számítógépekkel való kulcsfeltöréssel. Mióta felváltottam a kéttényezős hitelesítéssel, sokkal nyugodtabban alszom.
A kereskedelmi forgalomban kapható, hálózati alapú, kéttényezős hitelesítési rendszerek általában túl drágák és bonyolultak ahhoz, hogy azokat otthon vagy kis hálózatokban használjuk. De nem kell aggódni, van megoldás. A ***Linux*** már eleve tartalmazza a kéttényezős hitelesítéshez szükséges elemeket. A közismert, biztonságos kommunikációs eszköz, az ***OpenSSH***, minden olyan összetevőt tartalmaz, amely az otthoni számítógépekre, a kis hálózatokra és esetenként a nagyobb hálózatokra is megfelelő, gépalapú, kéttényezős hitelesítéshez szükséges. Ebben a cikkben megmutatom, hogy a hordozható eszközök, az ***OpenSSH*** nyilvános és titkos kulcsa, valamint a lenyűgöző ***ssh-agent*** kombinálásával hogyan lehet kéttényezős hitelesítést kialakítani úgy az egyszerű, mint a privilégizált felhasználók számára.

Első példa – Kéttényezős hitelesítés USB meghajtóval

Kezdjük az egyszerű (adminisztrátori jogokkal nem rendelkező) felhasználókkal. Ekkor használhatjuk az ***SSH*** jól ismert nyilvános hitelesítését, egy kis trükkel kiegészítve. Ahelyett, hogy a titkos kulcsot a felhasználó ***.ssh*** alkönyvtárában tárolnánk, az ***USB***-meghajtóra mentjük.
A példa kedvéért vegyük a nem privilégizált bob felhasználót, aki egy machine1 nevű, ***Fedora Core*** számítógépre jelentkezik be. A `machine2` nevű, távoli ***Linux*** gépre is ezzel a felhasználónévvel jelentkezünk be. Hozzuk létre az ehhez szükséges nyilvános és titkos kulcsokat:

```
ssh-keygen -t rsa -f key-rsa-
➥bob@machine2 -C key-rsa-
➥bob@machine2
```

A (lehetőség szerint minél hosszabb és véletlenszerűbb) generátorszöveg begépelése után, az ***ssh-agent*** létrehozza a kulcspárt, alapértelmezés szerint a felhasználó ***.ssh*** alkönyvtárában, jelen esetben a ***/home/bob/.ssh***-ban. A fájl neve tetszőleges lehet, de a példa kedvéért most olyan beszédes nevet választottam, amelyből egyetlen pillantással megállapítható a felhasználó és a gép neve – ennek jelentősége majd az ezt követő, több kulcsos példákban lesz érezhető. (Feltételezzük, hogy az ***USB*** meghajtó valamilyen linuxos, például ***ext3*** vagy ***vfat*** fájlrendszerrel formázott, de a kulcs fájljogosultságát minden beillesztésnél 400-ra kell állítani.)
Végezzük el az ***USB*** meghajtó beillesztését a fájlrendszerbe, miután azt ***/media/usbdisk*** (a példában ezt használjuk), ***/media/usbdisk1***, ***/media/disk*** vagy ***/media/disk-1*** néven kell látnunk. Az imént készített titkos kulcsot mozgassuk egy erre a célra szentelt könyvtárba és korlátozzuk a hozzáférési jogot a tulajdonosra:

```
mv key-rsa-bob@machine2
➥/media/usbdisk
chmod 400 /media/usbdisk/
➥key-rsa-bob@machine2
```

Ezután másoljuk be a nyilvános kulcsot (`key-rsa-bob@machine2.pub`) a `machine2`-n lévő ***/home/bob/.ssh/authorized_keys*** fájlba. Gondoskodjunk róla, hogy az ***authorized_keys*** fájlt csak a tulajdonosa olvashassa:

```
chmod 400 authorized_keys
```

Végül bejelentkezhetünk a `machine2` távoli gépre bob néven a létrehozott kulcspárral (az `ssh` program a `-i` kapcsolóból tudja, hogy melyik kulcsot kell használnia):

```
ssh -i /media/usbdisk/key-rsa-
➥bob@machine2 bob@machine2
```

A generátorszöveg begépelése után a `machine2`-n futó ***SSH***-kiszolgáló belépteti bobot. Szüntessük meg az ***USB***-meghajtó (vagy más hordozható eszköz) beillesztését a machine1 gépen, és a titkos kulcs máris biztonságban van. Ezzel megvalósítottuk a kéttényezős hitelesítést: az egyik tényező az ***USB***-meghajtó, amely a titkos kulcsot, a másik tényező pedig a fejünk, amely a generátorszöveget tárolja. A nyilvános kulcsú ***SSH***-hitelesítés bizonyára sokak számára ismert és mindennapos, a titkos kulcs áthelyezése egy hordozható eszközre pedig egyszerű módja a tényezők fizikai szétválasztásának.

Második példa – Kéttényezős adminisztrátori hitelesítés ssh-agent segédprogrammal

Az első példában bemutattuk, hogyan lehet egy távoli gépre biztonságosan bejelentkezni úgy, hogy a hitelesítési tényezők szétválasztását ***USB***-minimeghajtóval oldottuk meg. Ez a megoldás jól működik nem privilégizált felhasználóknál, szemben az adminisztrátorokkal. Meg kell találnunk a módját, hogy root felhasználóként is bejelentkezhessünk.

Az egyik lehetséges és kézenfekvő megoldás az, hogy a távoli gép ***SSH***-kiszolgálójában engedélyezzük a root belépését közvetlenül a hálózatból. Sem kulcs, sem jelszó nem utazik a hálózaton, mégis megsértjük azt az ősi rendszeradminisztrátori szabályt, miszerint ez nem megengedhető. Nincs tehát más út, mint megtalálni annak a módját, hogyan jelentkezhetünk be először normál felhasználóként, majd eztán adminisztrátorként. Ismét az ***OpenSSH*** húz ki minket a bajból. Most is a nyilvános és titkos kulcsokat használjuk, egy kis beállítási trükkel kiegészítve. Először is, állítsuk be a távoli ***SSH***-kiszolgálót úgy, hogy a root felhasználó a belső ***loopback*** interfészen keresztül bejelentkezhessen, a külső hálózatból azonban ne. Másodszor, az `ssh-agent` segédprogramot úgy állítsuk be, hogy a távoli gép a helyi gépen tárolt kulcs lekérdezésével hitelesíthesse a root felhasználót. A leírtak a következő lépésekkel valósíthatók meg:

1. Készítsünk egy nyilvános és titkos kulcspárt a root felhasználónak.
2. Másoljuk a nyilvános kulcsot a ***root authorized_users*** állományába a távoli gépen.
3. A helyi gépen futtassuk az `ssh-add` segédprogramot, hogy a titkos kulcs a gyorsítótárba kerüljön.
4. Jelentkezzünk be egyszerű felhasználóként a távoli gépre `ssh`-val, az első példában leírtak szerint, de ez alkalommal használjunk az átirányítást.
5. A távoli gépen jelentkezzünk be rootként a ***localhost*** interfészen, minek hatására a távoli ***SSH***-kiszolgáló lekérdezi a helyi gépet és hitelesíti a root felhasználót.

Az `ssh-agent` pontosan a kívánt funkcionalitást biztosítja: lehetővé teszi, hogy távoli ***SSH***-kiszolgálók a helyi gép gyorsítótárában lévő, megfejtett titkos kulcsok lekérdezésével hitelesítsenek felhasználókat. A kulcsok soha nincsenek továbbítva a két gép között – a titkos kulcsok a helyi munkaállomáshoz csatlakoztatott hordozható eszközön maradnak.

Az ***ssh-agent*** nagyon sokoldalú eszköz, de a beállítása nem feltétlenül triviális. Először is, meg kell fejteni a titkos kulcsot az `ssh-add` eszközzel, és át kell adni az `ssh-agent` programnak. Másodszor, meg kell mondanunk, hogy az `ssh-add` és az `ssh-agent` hogyan tud szót érteni egymással. Ez utóbbi egy ***socket***, melynek helye az `SSH_AUTH_SOCK` környezeti változóban található. Az `ssh-agent` alapértelmezés szerint önkényesen választ nevet a socketeknek, így az `SSH_AUTH_SOCK` korrekt beállítása nehézkes lehet.

A legtöbb disztribúció, így a ***Fedora Core*** is, automatikusan létrehozza az ***ssh-add*** és ***ssh-agent*** közötti kapcsolatokat a grafikus bejelentkezéskor (például ***GNOME*** és ***KDE***). Jelentkezzünk be parancssorban, majd adjuk ki a következő utasítást:

```
ssh-add -l
```

Ha az ***ssh-add*** és az ***ssh-agent*** tudnak egymással kommunikálni, akkor ennek vagy a nyilvános kulcsok listája, vagy a `"The agent has no identities"` üzenet lesz az eredménye, ha egy kulcs sem létezik.

Ha az ***ssh-agent*** bármi okból nem fut, vagy az `SSH_AUTH_SOCK` környezeti változó nincs megfelelően beállítva, a `"Could not open a connection to your authentication agent"` üzenetet kapjuk. Az utóbbi esetben hajtsuk végre a következő parancsot:

```
eval `ssh-agent`
```

Ez elindít egy `ssh-agent` példányt, és a környezeti változókat is megfelelően beállítja az aktuális parancsértelmezőben. Ezután az 1. példában megismert módon kulcspárt generálunk a rootnak:

```
ssh-keygen -t rsa -f key-rsa-
➥root@machine2 -C "key-rsa-
➥root@machine2"
```

Helyezzük a titkos kulcsot a hordozható eszközre, és adjunk a tulajdonosnak olvasási jogot, rajta kívül azonban senkinek semmit:

```
mv key-rsa-root@machine2
➥/media/usbdisk
chmod 400 /media/usbdisk/
➥key-rsa-root@machine2
```

Másoljuk a nyilvános kulcsot a `machine2` távoli gép */**root**/.**ssh**/**authorized_keys*** állományába. A következő paranccsal adjuk meg a root titkos kulcsát az `ssh-agent` segédprogramnak:

```
ssh-add -t 300 /media/usbdisk/
➥key-rsa-root@machine2
```

A generátorszöveg begépelése után, az ssh-agent az „Identity added: key-rsa-root@machine2 (key-rsa-root@machine2)” üzenettel tér vissza, ha a kulcsot megadta. (A `-t` kapcsoló szabályozza a kulcsok gyorsítótárban való elérhetőségének idejét, ami a jelen esetben 300 másodperc, azaz 5 perc. Ennek hiányában a kulcsok örökre elérhetők lesznek.)

Jelentkezzünk be a távoli számítógépre egyszerű felhasználóként:

```
ssh -A -i /media/usbdisk/
➥key-rsa-bob@machine2
```

A generátorszöveg megadása után, már be is léptünk a `machine2` távoli gépre. (Ez a parancs ugyanaz, mint amit az első példában láthattunk, de a `-A` kapcsolót használjuk átirányításra.)

A távoli gépen hajtsuk végre az `ssh-add -l`

parancsot, mire a root kulcsát kell megpillantanunk, amit az imént adtunk meg az `ssh-agent` segédprogramnak, például:

```
   2048 fa:5c:4b:73:88:26:
➥..:... /media/usbdisk/
➥key-rsa-root@machine2 (rsa)
```

A `su` paranccsal váltsunk át a root felhasználóra (a `machine2` gépen), és állítsuk be az ***SSH***-kiszolgálót úgy, hogy a ***loopback*** interfészen a root bejelentkezhessen. Ehhez a */**etc**/**ssh**/**sshd_config*** fájlban a következő módosítások szükségesek:

```
PermitRootLogin yes
AllowUsers bob@*
AllowUsers root@localhost.*
```

(Előfordulhat, hogy a ***loopback*** interfész címét numerikus formában kell megadni:

```
AllowUsers root@127.0.0.1.)
```

Mentsük el a beállításokat, és indítsuk újra az ***SSH***-kiszolgálót:

```
service sshd restart
```

A root felhasználóval jelentkezzünk ki, majd az ***OpenSSH*** segítségével ismét jelentkezzünk be:

```
ssh root@localhost
```

A `machine2` távoli gépen futó ***OpenSSH***-kiszolgáló most már fogadja az adminisztrátori bejelentkezéseket a ***loopback*** interfészen, de a külső hálózatból továbbra sem. A root hitelesítéséhez szükséges információkat a machine1 géppel tárgyalja meg. Figyeljük meg, hogy a root titkos kulcsa nem hagyta el a machine1 számítógépet!
Az ***OpenSSH***-t ily módon használva, hatékonyan kiválthatjuk a `su` (***switch user – felhasználóváltás***) és `sudo` segédprogramokat.
De még nem végeztünk teljesen! A biztonság tovább növelhető, ha a `su` parancs használatát kizárólag a helyben csatlakoztatott eszközökre korlátozzuk. Módosítsuk a */etc/pam.d/su* fájlt, hogy megakadályozzuk a `su` használatát a hálózaton keresztül:

```
auth        required
➥pam_securetty.so
```

Mostantól a `su` csak a parancssorból és a virtuális terminálokból használható.
Szüntessük meg az ***USB***-eszköz beillesztését és fizikai csatlakozását a géphez. Ahhoz, hogy most valaki megszerezze a titkos kulcsot, el kell lopnia az ***USB*** meghajtót. Még ha ez sikerülne is, akkor vagy a generátorszöveg megszerzése bizonyulna körülményesnek, vagy elképesztően nagy számítási teljesítményre volna szükség a kulcs feltöréséhez.

Harmadik példa – Feszítsük tovább a húrt

Mielőtt újdonsült rendszerünket kitesszük a „vadon” megpróbáltatásainak, nem árt, ha betömünk még egy biztonsági rést.
Ha az ***ssh-agent*** segédprogramot átirányítással használjuk, az a távoli ***SSH***-kiszolgáló számára lehetővé teszi a helyi számítógépen tárolt, titkos kulcs lekérdezését. Ha azonban így több számítógépre szeretnénk belépni, egy rosszindulatú felhasználó megbabrálhatja a kulcsokat az egyik gépen úgy, hogy beléphessen egy másikra. Ebben az esetben rosszabb a helyzet, mintha statikus jelszót használnánk.
A probléma bemutatásához bővítsük ki a példahálózatot a `machine3` géppel. Elkészítjük a kulcsokat bobnak és rootnak a `machine3`-ra, az első és második példa szerint, majd a root titkos kulcsát megadjuk a `machine1`-en futó `ssh-agent` segédprogramnak.
Ekkor `bob` `ssh`-val belép a `machine3`-ra, a `-A` átirányító kapcsolót is használva. Ha most végrehajtjuk az

```
ssh-add -l
```

parancsot, a `machine2` és `machine3` gépek nyilvános kulcsait látjuk:

```
2048 fa:5c:4b:73:88:...: ...
➥/media/usbdisk/key-rsa-
➥root@machine2 (RSA)
2048 26:b6:e3:99:c1:...: ...
➥/media/usbdisk/key-rsa-
➥root@machine3 (RSA)
```

A példában a `machine1`-en futó `ssh-agent` a másik két gép magánkulcsait a gyorsítótárba tölti. Ez az az `ssh-agent`, amely lehetővé teszi, hogy adminisztrátorként lépjünk be az utóbbi két gép bármelyikére, és amely sajnos azt is megengedi, hogy a `machine2` gépről valaki rootként belépjen a `machine3`-ra, és fordítva. Ez nincs rendjén.
Szerencsére, ez a hiányosság kiküszöbölhető a `-c` kapcsolóval. A biztonság fokozható, ha minden egyes távoli számítógép adminisztrátori kulcsának tárolásához külön `ssh-agent` példányt futtatunk. A -c kapcsoló jelzi az `ssh-agent` számára, hogy a felhasználónak jóvá kell hagynia minden olyan kulcs felhasználását, amely a gyorsítótárban van. A távoli gépek számára dedikált `ssh-agent` példányok a még ismeretlen biztonsági hibák esetén is garantálják, hogy egy gép kulcsa teljesen független marad a többiétől.
Az `ssh-add` jóváhagyási funkciójának használata egyszerű: minden kulcs megadásánál használjuk a `-c` kapcsolót. Tegyünk egy próbát. Indítsunk egyszerre két `ssh-agent` segédprogramot a `machine1`-en, előre definiált socketek megadásával:

```
ssh-add -c /media/usbdisk/
➥key-rsa-root@machine2
ssh-add -c /media/usbdisk/
➥key-rsa-root@machine3
```

Mostantól kezdve jóvá kell hagyni a kulcs felhasználását, ha `ssh`-val be akarunk lépni a `machine2` vagy `machine3` gépre.
Arra is lehetőség van, hogy külön `ssh-agent` példányokkal tároljuk az egyes kulcsokat. Ehhez indítsunk két `ssh-agent` segédprogramot a machine1-en, előre definiált socketek megadásával:

```
ssh-agent -a /tmp/ssh-agent-
➥root@machine2
ssh-agent -a /tmp/ssh-agent-
➥root@machine3
```

Ismét hangsúlyozom, hogy az általam használt elnevezések önkényesek, ugyanakkor a példák szempontjából rendkívül beszédesek. Állítsuk be a környezeti változót és adjuk meg a kulcsot a `machine2`-nek:

```
export SSH_AUTH_SOCK=/tmp/
➥ssh-agent-root@machine2
ssh-add -c /media/usbdisk/
➥key-rsa-root@machine2
```

Ismételjük meg a fenti lépéseket a `machine3`-ra is, a `machine2`-re vonatkozó indexek értelemszerű cseréjével:

```
export SSH_AUTH_SOCK=/tmp/
➥ssh-agent-root@machine3
ssh-add -c /media/usbdisk/
➥key-rsa-root@machine3
```

Most jelentkezzünk be a `machine3`-ra (mivel az `SSH_AUTH_SOCK` környezeti változót legutóbb ennek a gépnek az `ssh-agent` programjára állítottuk):

ssh-add

Az `ssh-add` segédprogrammal tilthatjuk vagy jóváhagyathatjuk a titkos kulcsok használatát. A tiltás bekapcsolása a `-x`, kikapcsolása a `-X` kapcsolóval lehetséges. A kulcs tiltásánál megadott jelszót kell használnunk a tiltás kikapcsolásához. A `-c` kapcsoló esetén az `ssh-add` minden alkalommal figyelmeztet, amikor az `ssh-agent` segédprogramhoz kérés érkezik egy kulcs használatára. A figyelmeztető üzenet az `ssh-agent` segédprogramot futtató gépen jelenik meg, és hatékonyan akadályozza meg a jogosulatlan felhasználókat abban, hogy hozzáférjenek a kulcsainkhoz.

Kettő vagy 2.X tényező

A helyben tárolt ***SSH***-kulcsokat és a hozzájuk tartozó generátorszövegeket egyesek külön tényezőkként kezelik, nem minden alap nélkül. Mégis, sokkal megnyugtatóbbnak érzem, ha a kulcsok tárolása fizikailag is elkülönül a számítógéptől. A kulcsok hordozható eszközön tárolása csökkenti annak lehetőségét, hogy valaki megszerezze és feltörje azokat. Fontos annak megértése, hogy a kulcsok tárolása például ***USB***-minimeghajtón nem küszöböli ki teljesen annak lehetőségét, hogy egy rosszindulatú felhasználó megkaparintsa azokat. Amíg a hordozható eszköz beillesztve van, a kulcsok veszélyben lehetnek, így a helyi gépen is megfelelő óvintézkedéseket kell tenni annak érdekében, hogy a bejelentkezés a távoli gépekre biztonságos legyen. Használjunk erős jelszót a helyi (parancssori) belépéshez, legyenek telepítve a legfrissebb biztonsági javítócsomagok stb. Összességében tehát nyilvános kulcsú hitelesítés használatával jobban járunk, mint a statikus jelszavakkal, de csak addig, amíg a munkaállomáshoz kellően nehéz hozzáférni. Aztán hogy ki mennyire szereti magát biztonságban érezni, azt már az egyéni beállítottság és persze az üldözési mánia súlyosságának foka határozza meg.

```
ssh -A -i /media/usbdisk/key-
➥rsa-bob@machine2 bob@machine3
```

Hajtsuk végre az

```
ssh-add -l
```

parancsot, hogy lássuk, mely kulcsok elérhetők a machine1-en. Örömmel tapasztaljuk, hogy csak a `machine3` rootjának kulcsa jelenik meg. Jelentkezzünk ki a `machine3`-ról, állítsuk át a környezeti változót a `machine2` `ssh-agent` programjára, és jelentkezzünk be a `machine2`-re:

```
export SSH_AUTH_SOCK=/tmp/ssh-
➥agent-root@machine2
ssh -A -i /media/usbdisk/key-
➥rsa-bob@machine2 bob@machine2
```

Ismét ellenőrizzük az elérhető kulcsokat:

```
ssh-add -l
```

A listák csak és kizárólag az adott gép adminisztrátori kulcsát fedik fel előttünk. Az előző példában használt egyetlen `ssh-agent` esetén mind a `machine2`, mind a `machine3` gép kulcsát láttuk volna.
Ha minden olyan géphez, amelyre be akarunk jelentkezni, külön `ssh-agent` példányt futtatunk, az kicsit több munkával jár.
Az `SSH_AUTH_SOCK` környezeti változó átállítása, mikor másik gépre akarunk bejelentkezni, finoman szólva macerás. A folyamat egyszerűsítéséhez, írtam egy ***tfssh*** (***two-factor ssh – kéttényezős ssh***) névre keresztelt parancsfájlt, melynek szintaxisa a következő

```
tfssh [username@]host [keydir]
```

A parancsfájl (letölthető a ***LinuxJournal FTP***-helyéről: ***ftp.ssc.com/pub/lj/listings/issue152/8957.tgz***) szükség szerint elindítja az ssh-agent példányokat, beállítja a környezeti változót, a root kulcsot megadja az `ssh-agent` példánynak és a megadott felhasználónévvel (username) bejelentkezik a távoli gépre (host). Megadható továbbá, hogy a kulcsokat melyik (keydir) könyvtárban keresse, és mennyi ideig tartsa azokat a gyorsítótárban.

Összefoglalás

A statikus jelszavak több bajt okozhatnak, mint amennyit használnak. Gátat kell szabnunk ezek további használatának, és terjesztenünk kell a kéttényezős hitelesítést, melynek eszközkészletét a sokoldalú ***OpenSSH*** biztosíthatja. A nyilvános és titkos kulcsok, az átirányítási funkció és a hordozható eszközök használatával az ***OpenSSH*** gondoskodik a kulcsok biztonságáról, mely egyszerű, olcsó és hatékony eszközt jelent a gép alapú, kéttényezős hitelesítési rendszer készítéséhez.

A kulcsok tárolása

A kulcsok tulajdonképpen bármely hordozható eszközön tárolhatók. A példákban ***USB***-minimeghajtót használtunk, mert kicsi, könnyű és egyszerűen kezelhető. Bátran használjunk újraírható ***CD-ROM*** vagy ***DVD***-lemezt, de akár floppyt is, ha úgy tetszik.

Az ismertetett kéttényezős hitelesítési rendszer beállítása és használata némi munkát kíván, amelyet azonban bőségesen kompenzál az elért biztonsági szint. A `tfssh` parancsfájllal még ettől is megszabadulhatunk, vagyis a kéttényezős hitelesítést minden előnyével élvezhetjük, a járulékos teendők mellőzésével.

Linux Journal 2006., 152. szám

Paul Serry
Több mint 20 éve UNIX és Linux rendszeradminisztrátor. Számos Linux-könyv, pl. a Network Linux Toolkit és a Knoppix for Dummies szerzője. John „maddog" Hall társszerzőjeként több Red Hat Linux for Dummies és Fedora Core for Dummies könyvet is írt. Az új mexikói Albuquerque-ben él, elektronikus levélcíme pgsery@swcp.com.

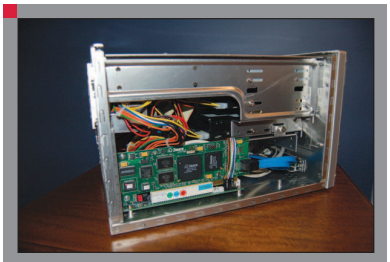
Egy megbízható, automatikus adatmentési megoldás

Ebben a cikkben bemutatom, miként alakíthatunk ki felügyelet nélkül működő, titkosított, redundáns elemekből álló hálózati adatmentési rendszert Linux operációs rendszerrel, a Duplicity szoftverrel, s mindezt közönséges, kereskedelmi forgalomban kapható hardverelemekből.

Manapság teljesen általános, hogy a felhasználók óriási merevlemezeket használnak, amelyeket persze dugig töltenek filmekkel, zenékkel, digitalizált videókkal, szoftverekkel, dokumentumokkal, és mindenféle más formátumú adattal. Ráadásul sokan teljesen elhanyagolják a *CD*-re és *DVD*-re történő adatmentést, aminek lélektanilag talán az lehet a legfőbb oka, hogy ennél az adatmentési megoldásnál rengeteg apró tennivalója akad a felhasználónak. Szinte biztosan meg kell birkóznia például olyan problémákkal, mint az adathordozó méretéből adódó korlátok „kerülgetése”, vagy az adatok épségének ellenőrzése. És akkor a manuális lemezcserélgetést még nem is említettem. Ennek megfelelően ezeknek az adatoknak a többségét nem is mentik a tulajdonosok, vagy legalábbis nem rendszeresen. Jómagam biztonsági szakértőként dolgozom, legfőképpen a szoftverfejlesztés területén, szabadidőmben pedig nyílt forrású szoftvereket fejlesztek. Ez utóbbi munkám keretében immár számos projektben vettem részt. Tekintettel meglehetősen széles érdeklődési területemre otthon egy 12 gépből álló hálózatom van, amelynek tagjain ***Linux***, ***Mac OS X*** és ***Windows*** is fut.
Az elmondottak alapján talán érthető, hogy a munkám eredményének megsemmisülése számomra nem elfogadható alternatíva. Ahhoz, hogy az én munkakörnyezetemben egy adatmentési rendszer valóban jól működhessen, több különböző gépen dolgozó felhasználót, és persze többféle operációs rendszert kell támogatnia. Valamennyi felhasználónak képesnek kell lennie a számára fontos adatok mentésére, illetve szükség esetén az önálló visszaállítására. Röviden tehát a rendszernek képesnek kell lennie a többfelhasználós, felügyelet nélküli működésre. Ha egy ilyen rendszer jól van kialakítva, akkor a felhasználó nem csak egy teljes mentést tud szükség esetén visszaállítani, hanem akár egyes fájlokat is, függetlenül attól, hogy azok mikor kerültek az archívumba. A többfelhasználós működés miatt nyilván szükség lesz egy megfelelően átgondolt biztonsági rendszerre is. Ennek része kell legyen a hozzáférés szabályozása, illetve az adatintegritás felügyelete is. Az előbbi abban akadályozza meg a felhasználókat, hogy egymás bizalmas adataihoz illetéktelenül hozzáférjenek, arról másolatot készítsenek, míg az utóbbi azt hivatott biztosítani, hogy visszaállításkor az adatok valóban a mentéskor érvényes állapotukban kerüljenek vissza rendeltetési helyükre.
A biztonság mellett egy adatmentési rendszernek a megbízhatóság is igen lényeges tulajdonsága. A megoldásnak ennek szellemében tolerálnia kell a hardver esetleges meghibásodásait. Mivel egy adattárolási rendszer legnagyobb valószínűséggel meghibásodó eleme a merevlemez, gondoskodnunk kell a megfelelő hibatűrésről. Végezetül a megoldásnak hatékonyan kell kihasználnia a rendelkezésre álló tárterületet, illetve hálózati sávszélességet. A sávszélesség ügyes elosztásával egyszerre több felhasználó férhet hozzá a rendszer szolgáltatásaihoz, míg a tárterülettel való hatékony gazdálkodás eredménye nyilván a tárolható adatok mennyiségében fog megmutatkozni. Mint minden munkámmal kapcsolatban, természetesen itt is törekedtem arra, hogy a végeredmény vizuálisan is vonzó illetve kellően kicsi legyen, és persze az ára is az „elfogadható” kategóriába essen.
Először azzal próbálkoztam, hogy egy már létező megoldást találjak, amit készen kapok. Találtam is számos jelöltet, amelyek nagyjából két kategóriába sorolhatók: vannak egylemezes hardveres hálózati adatmentési megoldások, és léteznek ugyanilyen ***RAID*** tömbök. Az első kategória egyik maga nemében kiváló képviselője a ***Western Digital NetCenter*** nevű terméke. Ugyanakkor hamar felismertem, hogy egyetlen, ebbe a csoportba tartozó megoldás sem rendelkezik azokkal a tulajdonságokkal, amelyeket az imént felsoroltam. Nincsenek vagy hiányosak a biztonsági szolgáltatások, rossz a sávszélesség kihasználása, vagy egyszerűen csak nem kellően megbízható a termék. A második csoportba tartozó eszközöket láthatóan inkább céges, sem mint magáncélú

1. ábra A Silver Venus 668 ház (elölnézet)

2. ábra A Silver Venus 668 ház (hátsó nézet)



3. ábra A Silver Venus 668 ház (belülről a már beszerelt hardverelemekkel)

felhasználásra tervezték. Ennek egyik kellemetlen mellékhatása, hogy általában jóval drágábbak, mint az első csoport tagjai. A ***Snap Server 2200*** kiváló példa ennek a kategóriának az alacsony árfekvésű részére. Az eszköz ára 1000 dollártól indul, viszont tény, hogy tekintélyes nagyságú tárterület alakítható ki vele. Ezzel együtt a drága készülékekről is azt voltam kénytelen megállapítani, hogy csak hellyel-közzel tesznek eleget a biztonsággal, teljesítménnyel, vagy egyéb szolgáltatásokkal kapcsolatos általános elvárásaimnak.
Mivel a keresés végeredménye az volt, hogy a könnyen és gyorsan elérhető megoldások között egyetlen számomra megfelelő sincsen, úgy döntöttem, hogy magam fogok megépíteni egy ilyen készüléket.
A rendszeremnek biztonságosnak és megbízhatónak kell lennie, titkosítást kell használnia, Linux operációs rendszerrel kell működnie, és végül, de nem utolsó sorban olyan elemekből kell állnia, amelyek közönséges kereskedelmi forgalomban is kaphatók (***Commercial Off-The-Shelf***; ***COTS***).
Azt már az elején eldöntöttem, hogy magát az adatmentést és kezelést a ***Duplicity*** nevű csomaggal fogom megoldani. Ezekkel az eszközökkel és elvárásokkal végül is valóban sikerült megépítenem egy olyan hálózati eszközt, amely egyaránt képes teljes és inkrementális mentéseket készíteni, az adatokat pedig digitális aláírással titkosítva tárolja. Az inkrementális mentés olyan adatmentési megoldás, amelynél csak a legutolsó mentés óta megváltozott tartalmak kerülnek be az aktuális mentésbe.
A teljes mentésnél ezzel szemben az eredeti adathordozó teljes tartalmáról másolatot készítünk, függetlenül az egyes fájlok korától. Ami a visszaállítást illeti, a rendszerem a teljes adatvisszaállítás mellett képes arra is, hogy megadott fájlok egy megadott időben készült másolatát visszaírja az eredeti adathordozóra. Utóbbira számos esetben lehet szükségünk. Tegyük fel például, hogy kaptam egy vírust, de azt is pontosan tudom, hogy a kártevő egy hete még nem volt ott a rendszeremen. Az egyedi adatmentési megoldásommal simán megtehetem, hogy visszaállítom a megfertőződött rendszer egy héttel, egy hónappal korábbi állapotát, vagy akár a legelső mentéskor érvényeset.
A ***Duplicity*** a projekt hivatalos weblapja szerint egy olyan adatmentési megoldás, amely teljes könyvtárakról készít tar formátumú, titkosított mentéseket, majd azokat feltölti egy helyi vagy távoli fájlkiszolgálóra.
Az általam elkészített adatmentési megoldás sarokköve is ez az alkalmazás volt. A ***Duplicity*** támaszkodik a ***librsync*** és a ***GnuPG*** könyvtárakra, valamint számos más fájlátviteli mechanizmusra. Ez volt az a rendszer, amely rendelkezett mindazokkal a képességekkel, melyek az általam megálmodott megoldás funkcionalitásának biztosításához elengedhetetlenek voltak. Volt benne biztonság, hatékonyan működött, egyszerűen mindent tudott, amit csak akartam.
A ***Duplicity*** először a ***librsync*** segítségével elkészít egy tar formátumú kötetet, amely vagy egy teljes, vagy egy inkrementális mentést tartalmaz. Azután ezt az adattömböt a ***GnuPG*** segítségével titkosítja és digitálisan aláírja, ami egyszerre biztosítja az adatok integritását és idegen számára való hozzáférhetetlenségét. Amikor készen van a titkosított mentés, a ***Duplicity*** a megadott helyre továbbítja azt számos adatátviteli mechanizmusainak egyikével.
Jómagam az ***SSH***-n keresztül történő fájlátvitelt használtam, mivel az adatok így a másolás során is titkosított formában áramlanak. Erre tulajdonképpen már nem is lenne szükség, hiszen maguk az átvinni kívánt adatok eleve titkosítottak, viszont a biztonságos átvitel még egy szinttel nagyobb összetettséget kölcsönöz a rendszernek, ami egy esetleges támadó számára értelemszerűen újabb akadály. A döntésnél persze az is lényeges szempont volt, hogy ***SSH*** gyakorlatilag minden gépen fut, így nem kell újabb hálózati szolgáltatásokat, például ***FTP***, ***NFS*** vagy ***rsync*** szervert beüzemelni.

A hardver

Miután eldöntöttem, hogy belevágok, és megépítem magamnak ezt a hálózati adatmentő eszközt, a következő lépésben el kellett döntenem, hogy milyen hardverelemekből fogok építkezni. Figyelembe véve a megvalósítani kívánt funkciókat, valamint a megbízhatósággal, a biztonsággal és a teljesítménnyel kapcsolatos elvárásaimat, világos volt, hogy egy ***RAID 1***-es – tükrözött – tömbön alapuló hálózati megoldást kell megépítenem. Mindez azt jelentette, hogy két merevlemezre volt szükségem, meg persze egy olyan ***RAID*** kártyára, ami legalább két meghajtót képes vezérelni.
Ami az alaplapot illeti, ott rögtön a kis formafaktorú változatok körül kezdtem el nézelődni. Korábban számos esetben használtam már ***Mini-ITX*** rendszerű alaplapokat, így pontosan tudtam, hogy ezek linuxos támogatottsága csaknem teljesnek mondható. Tekintettel arra, hogy ehhez az eszközhöz fölösleges lett volna valamilyen erős processzort választani, az ***EPIA Mini-ITX ML8000A***

alaplap mellett döntöttem, amelyen egy 800 MHz-es processzor, egy 100 Mb-es hálózati csatoló és egy 32 bites *PCI* foglalat található. Ezek a paraméterek tökéletesen megfeleltek a számítási sebességgel és hálózati átvitellel kapcsolatos elvárásaimnak, és ott volt a *PCI* csatoló is a *RAID* kártyának.
Megvolt tehát a megfelelő formafaktorú alaplapom, így választanom kellett egy hozzá illő házat és tápegységet. Itt nyilván fontos szempont volt, hogy a házban a *Mini-ITX* alaplap mellett azért el kell férnie a *RAID* vezérlőnek, meg két teljes méretű merevlemeznek is, miközben az általános esztétikai igényeimről sem feledkezhetek meg. Nos, a választás itt nehéznek bizonyult. Egészen sok *Mini-ITX* ház tulajdonságait böngésztem át, mire ráakadtam az igazira, amely a maga nemében egyetlennek is bizonyult. A nagy ő a *Silver Venus 668* lett, amely kellően sokrétű ahhoz, hogy minden igénynyemnek megfeleljen. Megvolt tehát az alaplap és a ház. A következő lépés a memória kiválasztása volt. Úgy döntöttem, hogy 512 MB DDR266-os *RAM* tökéletesen elegendő lesz.
Azért végül mégis akadt egy kis problémám: furcsa kimondani, de nem volt éppen egyszerű *Mini-ATX* alkatrészeket forgalmazó céget találni az *Egyesült Államokban*. Aztán végül mégis találtam egyet, a *Logic Supply* nevűt, amelynél az alaplapot, a házat és a memóriát is megtudtam rendelni, összesen 301,25 dollárért, amiben már a szállítási költség is benne volt. Ezen a ponton tehát a kezemben volt minden alkatrész, kivéve a *RAID* vezérlőt.
És itt jött az igazi probléma. A minden tekintetben megfelelő *RAI*D kártya megtalálása kifejezetten nehéznek bizonyult. Először is számos olyan *RAID* vezérlő van, amelyik a munka dandárját nem hardverből valósítja meg, hanem az operációs rendszer által futtatott meghajtóval végezteti el. A választás végül *3ware 8006-2LP SATA RAID* vezérlőre esett, amely két *SATA* vezérlővel rendelkezik, a vezérlést pedig teljesen a kártyán levő chipek végzik. Ezt az eszközt a *Monarch Computer Systems*-től vettem meg 127,83 dollárért (az ár ismét tartalmazta a szállítás költségét is).

Most már tényleg csak a merevlemezek hiányoztak. Némi töprengés után végül két 200 GB-os *Western Digital #2000JS SATA300* márkajelű lemezt vettem, melyek 8 MB cachesel rendelkeznek. Ezeket a *Bytecom Systems Inc*-től rendeltem meg, szállítással együtt összesen 176,69 dollárért. Megvolt tehát minden hardverelem a rendszer megépítéséhez a végösszeg pedig 604,77 dollárra rúgott. Összehasonlításként a legegyszerűbb készen kapható *RAID* vezérlővel szerelt hálózati adatmentő egységek 1000 dollárnál kezdődnek, ráadásul nem is nyújtanak annyit, mint amennyit az én rendszerem fog tudni.

A fájlkiszolgáló

Miután megépítettem magát a gépet, el kellett döntenem, hogy milyen operációs rendszer fut majd rajta. Választásom a *Debian stable 3.1r2* terjesztésre esett, elsősorban a kiváló csomagkezelés miatt. Telepítettem egy *SSH* démont is, hiszen ezen keresztül lehet majd elérni a kiszolgálót. Miután ezzel is megvoltam, létrehoztam magamnak egy felhasználói fiókot. A mentett adatok minden felhasználó saját könyvtárába fognak kerülni, vagyis mindenkinek, aki adatokat szeretne menteni a rendszerre, rendelkeznie kell rajta egy fiókkal.

Az ügyfelek beállítása

A kiszolgáló ezzel tulajdonképpen üzemkész is volt, a következő lépésben tehát a hálózat többi gépét kellett megfelelően beállítanom. Mivel a *Duplicity* – mint korábban is említettem – a *GnuPG*-re és az *SSH*-ra támaszkodik, ezeket a szolgáltatásokat úgy kell üzemeltetni, hogy rendszergazdai beavatkozás nélkül legyen képesek együttműködni a *Duplicity*-vel. A következő szakaszokban ismertetem azt a konfigurációt, amit a hálózati adatmentési kiszolgálót használó gépeken valósítottam meg.

A Duplicity telepítése

A *Duplicity* rendszert egyszerűen a *Debian Linux* `apt-get` parancsával telepítettem rendszergazdaként a következőképpen:

```
# apt-get install duplicity
```

Hitelesítés SSH kulcsokkal

Amint telepítettem a *Duplicity*-t, létrehoztam egy *DSA* kulcspárt, és úgy állítottam be az *SSH*-t, hogy ezt használva hitelesítse a felhasználókat. Ebben a módszerben értelemszerűen az a jó, hogy bejelentkezéskor nem kell jelszót megadni. Ezen a ponton talán érdemes megjegyezni, hogy egyesek a kulcspárral való hitelesítést úgy használják, hogy magukat a kulcsokat sem védik jelszóval. Ez biztonsági szempontból rendkívül kockázatos, hiszen bárki, aki megkaparintotta a belépéshez használt kulcsot, attól kezdve ugyanolyan jogosultságokkal rendelkezik, mint mi magunk. A kulcsokat tehát mindenképpen érdemes jelszóval védeni, amit a használat megkezdése előtt a rendszer ellenőriz. Az *SSH DSA* kulcspár előállításához a következő parancsokat futtattam az ügyfélgépen:

```
$ ssh-keygen -t dsa
$ scp ~/.ssh/id_dsa.pub
➥<username>@<server>:
$ ssh <username>@<server>
$ cat id_dsa.pub >> ~/.ssh/
➥authorized_keys2
$ exit
```

A kulcspárt ténylegesen az első parancs állítja elő. A másodikkal a már kész nyilvános kulcsot átmásoljuk a mentéshez használt kiszolgálóra. A harmadikkal elindítunk a kiszolgálón egy távoli parancsértelmezőt, a negyedikkel pedig hozzáfűzzük a most előállított nyilvános kulcsot a használható kulcsok listájához. Ez az a lépés, amivel végül is engedélyezzük, hogy a két gép között a hitelesítés kulcspár használatával történjen. Végezetül az ötödik paranccsal kilépünk a távoli parancsértelmezőből.

A GnuPG kulcsok beállítása

Az *SSH* kulcsok előállítása és beüzemelése után még elő kellett állítanom egy olyan *GnuPG* kulcspárt is, amivel a *Duplicity* a mentett adatokat fogja titkosítani. Ezeket a kulcsokat közönséges felhasználóként, az ügyfélen hoztam létre. Mivel a titkosító kulcsok egy közönséges felhasználói fiókhoz tartoznak, nem lehet velük a teljes fájlrendszert lementeni.
Ha valaha mégis szükség lenne arra,

hogy teljes mentést tudjak készíteni, akkor sincs más dolgom, mint rendszergazdaként bejelentkezve előállítani egy másik *GnuPG* kulcspárt. A *GnuPG* kulcsok létrehozásához tehát a következő parancsot használtam.

```
$ gpg --gen-key
```

A kulcskarika

Amint a *GnuPG* és az *SSH* kulcsok egyaránt elkészültek, az első dolgom az volt, hogy írtam egy *CD*-t, amire kimásoltam valamennyit. Aztán telepítettem és beállítottam a *Keychain* nevű alkalmazást. Ez egy olyan program ami a hosszan futó *ssh-agent* és *gpg-agent* példányokat képes kezelni úgy, hogy ne kelljen megadni a kulcsokhoz tartozó titkosító jelszót minden egyes olyan program indításakor, amelynek szüksége van a kulcsokra. *Debian* előtt először telepítenem kellett a *keychain* és az *ssh-askpass* nevű csomagokat. Aztán a */etc/X11/Xsession.option* nevű fájlban ki kellett kommenteznem az *ssh-agent* kifejezést tartalmazó sort, hogy az ügynökprogram ne indul el, valahányszor elindítok egy új *X* munkamenetet az *Xsession* segítségével. Aztán a saját könyvtáramban található *.bashrc* fájlba beleírtam a következő néhány sort, hogy a *Keychain* megfelelően induljon el:

```
/usr/bin/keychain ~/.ssh/
➥id_dsa 2> /dev/null
source ~/.keychain/`hostname`
➥-sh
```

Végül a *gnome-session* beállításaim közé fölvettem egy *xterm* indítást. Az *xterm* automatikusan elindítja a *Bash* héjat, ez induláskor elolvassa a *.bashrc* tartalmát, és elindítja a *Keychain* alkalmazást. A *Keychain* induláskor ellenőrzi, hogy a kulcsok bekerültek-e már a gyorstárba. Ha nem, akkor egyetlen egyszer megkérdezi a kulcsot titkosító jelszót, valahányszor elindítom a gépemet és bejelentkezem.

A Duplicity használata

Ha a *Keychain* is tökéletesen működik, már nincs is más hátra, mint használatba venni az új rendszert. A *Duplicity* segítségével immár bármely könyvtáramról felügyelet nélkül biztonsági másolatot készíthetek az adatmentési kiszolgálóra úgy, hogy egy *cron* feladattal automatizálom a program indítását. Próbaképpen a teljes *home* könyvtáramról készítettem egy biztonsági másolatot a következő paranccsal:

```
$ duplicity --encrypt-key
➥AA43E426 \
--sign-key AA43E426 /home/
➥username \
scp://user@backup_serv/backup/
➥home
```

Miután lezajlott a mentés, a következőképpen ellenőriztem annak helyességét:

```
$ duplicity --verify --encrypt
➥-key AA43E426 \
--sign-key AA43E426 \
scp://user@backup_serv/backup/
➥home \
/home/username
```

Most tegyük fel, hogy az ügyfélgépen véletlenül letöröltem a *home* könyvtáram teljes tartalmát. Az adatmentési kiszolgálóról való visszaállításhoz a következőt kell tennem:

```
$ duplicity --encrypt-key
➥AA43E426 \
--sign-key AA43E426 \
scp://user@backup_serv/backup/
➥home \
/home/username
```

Igen ám, csakhogy normális esetben az ember a *home* könyvtárában tartja a *GnuPG* és *SSH* kulcsait is, amelyek nélkül ez a művelet sajnos nem fog menni. Először tehát fogom azt a bizonyos *CD*-t, amit rögtön a kulcsok előállítása után megírtam, visszaállítom róla a kulcsokat, és máris működik minden. A *Duplicity* segítségével akár azt is megtehetjük, hogy a kiszolgálóról töröljük egy-egy fájl bizonyos időpontban készült mentését. Ezt kihasználva jómagam a következő parancsot fölvettem a *crontab*-omba, amely minden, két hónapnál idősebb mentést automatikusan töröl a szerverről:

```
$ duplicity --remove-older-
➥than 2M \
--encrypt-key AA43E426 --sign
➥-key AA43E426 \
scp://user@backup_serv/backup/
➥home \
/home/username
```

Ezzel egyik oldalról helyet spórolok, másrészt azonban az intézkedés értelemszerűen korlátozza azt az időt, amennyire a rendszer segítségével „visszanézhetek” és visszaszerezhetem az elveszett adataimat.

Összefoglalás

A cikkben bemutatott adatmentési megoldás nálam a leghatározottabban bevált. Rendelkezik minden olyan funkcióval, ami számomra fontos, és összességében kielégíti valamennyi igényemet. Ugyanakkor azt is látni kell, hogy még ez a rendszer sem tökéletes. A *Duplicity* például egyelőre nem támogatja a közvetlen láncokat (*hard link*), helyette ezeket is közönséges fájlokként kezeli. Ennek megfelelően ha olyan anyagot mentünk, majd állítunk vissza, amelyben ilyen láncok is előfordultak, akkor redundancia keletkezik, hiszen a visszaállításnál egyedi fájlok, és nem hivatkozások jönnek létre.
E fogyatékossága ellenére számomra még mindig a *Duplicity* a legmegfelelőbb megoldás. Ráadásul úgy tűnik, hogy a fejlesztők újabban belelendültek a munkába, tehát lehetséges, hogy az imént említett kellemetlenséget is hamarosan kiküszöbölik. Meg aztán az is lehet, hogy én magam fogom megoldani ezt a dolgot, és közkinccsé teszem. Akárhogy is lesz, az biztos, hogy ezzel a rendszerrel olcsón és viszonylag kevés munkával meg lehet valósítani egy tetszőleges, felügyelet nélkül működő titkosított hálózati adatmentő kiszolgálót.

Linux Journal 2006., 153. szám

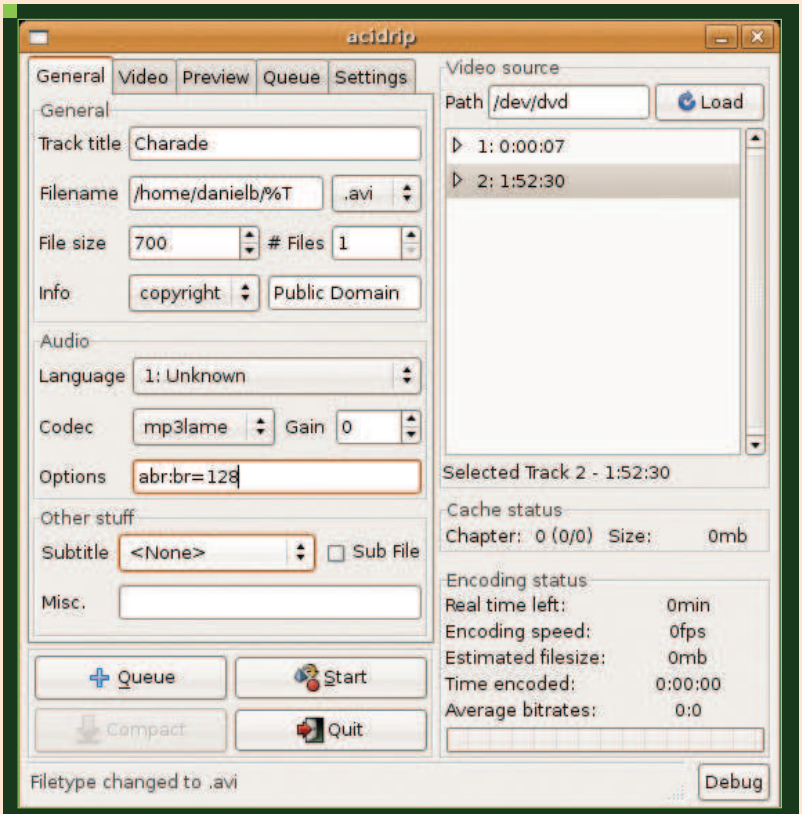
Andrew J. De Ponte biztonsági szakértő aki szoftverfejlesztéssel is gyakran foglalkozik. 1997 óta számos UNIX-változattal dolgozott már, és úgy gondolja, hogy a siker kulcsa az, ha az ember megtalálja az egyensúlyt a termelékenység és a szép tervezés között. Az esetleges kérdéseket és megjegyzéseket a cyphactor@socall.rr.com címen várja.

Egy Gtk2-es felhasználói felület a MEncoderhez –

Az AcidRip

A cikkből megtudhatjuk, hogyan használhatjuk az AcidRip felületet arra, hogy segítségével biztonsági másolatot készítsünk kedvenc DVD lemezeinkről...

A *MEncoder* egy kicsi de annál nagyszerűbb parancssori eszköz, amely az *MPlayer* csomag részét képezi, és alapvetően videó adatfolyamok kódolására való. Bemenetként bármilyen, az *MPlayer* által ismert videóformátumot megadhatunk neki. Ebbe tehát beleértendő a *Windows Media*, az *MPEG-2 (DVD)*, a *QuickTime*, az *MPEG-4*, a *DivX*, és még számos más médiaformátum. A program különböző kódolókkal (ilyen például a *lavc*, a *libdv*, az *xvid* vagy az *x264*) képes ezeket a formátumokat egymásba átalakítani.
Hogy miért adja a fejét valaki arra, hogy egy filmből előállítsa gyakorlatilag ugyanazt a filmet, csak épp más formátumban, annak számos különböző oka lehet. Az egyik ilyen cél például az, ha az *Észak-Amerikában* szabványos, 29,97 kép/másodperces vetítési sebességgel működő *NTSC* formátumból akarunk olyan filmet elállítani, ami az *Európában* használt *PAL* szabványnak, és a vele járó 25 képkocka/másodperces lejátszási sebességnek felel meg. De lehet a cél a filmkockákon található karcolások és pornyomok eltüntetése, vagy a színkorrekció is. Ami engem illet, én a szükséges tárhely csökkentése végett szoktam konvertálni, ugyanis az újabb kodekek, mint például az *xvid* vagy az *x264* kisebb hely felhasználásával nyújtanak gyakorlatilag többet, mint az olyan régi formátumok, mint teszem azt az *MPEG-2*. Ezzel a trükkel egy alapvetően 4 GB méretű *DVD* filmet a minőség érzékelhető romlása nélkül akár 2 GB területen is tárolhatunk. Ha pedig valaki úgy gondolja, hogy a cél szentesíti az eszközt, és ennek szellemében nem riad vissza a kép méretének csökkentésétől sem, az akár egyetlen *CD-ROM*-ra is rázsúfolhatja a filmet. Az igazság pedig az, hogy még ennél a lefokozott méretnél is kiváló hang és képi élményben lehet részünk, feltéve persze, hogy ügyesen tudjuk használni a *MEncodert*.
Sajnálatos módon azonban a *MEncoder* megfelelő használatát nem is olyan egyszerű megtanulni. Van persze súgóoldal hozzá: 7216 sor hosszú. Megfelelő türelemmel, és persze elegendő szabadidővel felfegyverkezve természetesen ezt is el lehet sajátítani, mint bármi mást, de én se türelmes nem vagyok, se időmilliomos. Hogy rövid legyek, a megoldandó problémám a következő volt: a gyerekeim valami rejtélyes oknál fogva elhatározták, hogy összetörik az összes, a lakásban található *DVD* lemezt. Persze nem direkt csinálják, egyszerűen csak gyerekek, de ez a végeredményen nem sokat változtat. Szó mi szó, gyereket *DVD*-vel keverni nem jó ötlet. Ehhez a jelenleg kapható lemezek sajnos túl törékenyek. Sőt, nem is kell hozzá föltétlen gyerek, hogy csúnya karcolások és kisebb felületi törések keletkezzenek a korongokon. Amint kinyitjuk a tokot, máris ki van téve a drága adathordozó mindenféle szörnyűségnek. A gyerekeim eddig – többek között – a következő filmeknek „jártak a végére”: *Shrek*, *Ice Age*, *Black Beauty* and *Chitty Chitty Bang Bang*. És persze ezzel a bűnlajstromuk még nem ért véget, de mint mondtam a gyerek az gyerek, tehát fölösleges a felsorolást folytatni. Viszont a rombolást ezen a ponton talán nem ártana megállítani.
A tervem az volt, hogy – megelőzve a csimotákat – összeszedem a lakásban az összes, még épségben megmaradt *DVD*-t, és archiválom őket a Linux kiszolgálómra. Ezzel a gyerekek se járnak rosszul, hiszen a *MythTV* vagy valamilyen másik, a célnak megfelelő frontend segítségé-



1. ábra Az AcidRib General Settings (Általános Beállítások) füle

vel a tévékészüléken keresztül továbbra is lejátszhatják a kedvenc filmjeiket, ahányszor csak akarják. Az eredeti lemezeket pedig szépen visszahelyezzük a tokokba, és egy kellően hozzáférhetetlen helyre elzárjuk őket.

A szerveremben egyelőre ugyan bőven volt hely, de ha belegondolok, hogy a tárolás lemezenként 4 GB-ot tesz ki, a gyűjteményünk pedig, amely már most is körülbelül 100 lemezből áll folyamatosan bővül, akkor kénytelen leszek eltöprengeni a tárolás hatékonyságának növelésén. És ez volt az a pont, ahol beugrott a ***MEncoder***. A megoldás nyilván az lesz, hogy ezzel a programmal átkódolom a filmeket ***MPEG-2***-ből valami olyan formátumba, ami kevesebb helyet igényel.

Szóval ez volt a cél és az elhatározás, de a megvalósítás azért elsőre nem tűnt egyszerűnek.

A ***MEncoder*** tervezőinek szemmel láthatólag az volt a célja, hogy a felhasználó számára lehetővé tegyék a filmek valamennyi tulajdonságának meghatározását. A kódolásnál beállíthatjuk a formátumot, a lejátszási sebességet, a bitsebességet, a méreteket, a színeket, és mindent, ami egyáltalán állítható. És ha az ember a kiváló eszközöknek ekkora tárházával gazdálkodhat, akkor bizony kiváló alkalma nyílik a legkülönbözőbb hibák elkövetésére is. Nagy erő, nagy hibák! Bizonyos jelek arra utalnak, hogy ezzel a problémával nem én találkoztam először. Mások is megszenvedték már a tanulási folyamat kínjait, és szerencsére olyan is akadt köztük, aki elhatározta, hogy a többieknek megkönnyíti a dolgot. Az eredmény egyelőre nem tökéletes, de mindenképpen felfogható úgy, mint a helyes irányba megtett első lépés.

Az ***AcidRip*** egy a ***MEncoderhez*** írt ***Gtk2::Perl*** frontend. Segítségével grafikus felületen adhatjuk meg a különböző kódolási paramétereket, a program pedig közben éberen figyel, és fölhívja a figyelmünket, ha a paramétereknek olyan kombinációját sikerült megadnunk, amiből minden lesz, csak élvezhető film nem.

Az ***AcidRip*** forrását a ***SourceForge*** megfelelő oldaláról tölthetjük le, de megtalálható számos ***Linux*** terjesztés csomaglistájában is. Mivel az ***AcidRip*** alapvetően egy ***Perl*** program, ha kibontottuk a csomagot, fordításra nincs szükség. Azonnal futtatható abból a könyvtárból, ahova a telepítés során került.

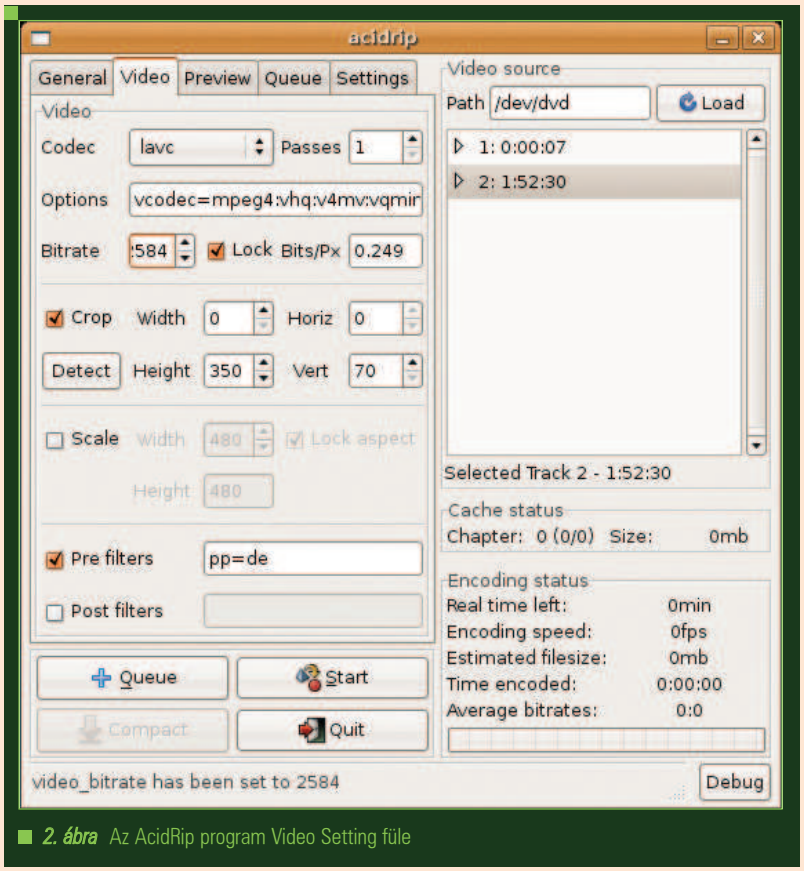
Az ***AcidRip*** – mi sem természetesebb – az ***MPlayerre*** és a ***MEncoderre*** támaszkodik, vagyis mielőtt használni kezdhetnénk, ezeket mindenképpen telepítenünk kell, és el kell végezni a futtatásukhoz szükséges beállításokat is. Szintén szükségünk lesz a ***DeCSS*** csomagra, ami a kódolt ***DVD***-k olvasásához kell. Összességében elmondható, hogy azt a ***DVD***-t, amit az ***MPlayerrel*** le tudunk játszani, archiválni is tudjuk a ***MEncoder*** segítségével. Mivel az ***MPlayer*** és a ***MEncoder*** is része a legtöbb ***Linux*** terjesztésnek, nagy rá az esély, hogy azonnal használható bináris csomagot is találunk a telepítőlemezeken, vagy a terjesztésünk kiegészítő anyagait tartalmazó webhelyen. Ha mégsem így lenne, akkor töltsük le a program forrását, illetve a szükséges kodekeket az ***MPlayer*** hivatalos webhelyéről, majd kövessük a fordítási és telepítési útmutatót (lásd az on-line forrásokat).

Az ***AcidRip*** szintén használ egy ***lsdvd*** nevű kicsi programot, de ez benne van a telepítési csomagjában.

Ha elindítottuk a programot, először is be kell töltenünk egy ***DVD*** lemez tartalmát. Helyezzünk be tehát egy korongot, majd nyomjuk meg a ***Video Source*** szakaszban található ***Load*** gombot. Erre megjelenik a ***DVD***-n található valamennyi fejezet és sáv. Én a példa kedvéért a továbbiakban azt feltételezem, hogy a lemez mindössze két fejezetet tartalmaz. Más lemezeken ennél természetesen több is lehet. Ha egyenként látni szeretnénk az egyes fejezetekben található sávokat, nyomjuk meg a sáv neve mellett található kis háromszög alakú ikont a lista kibontásához.

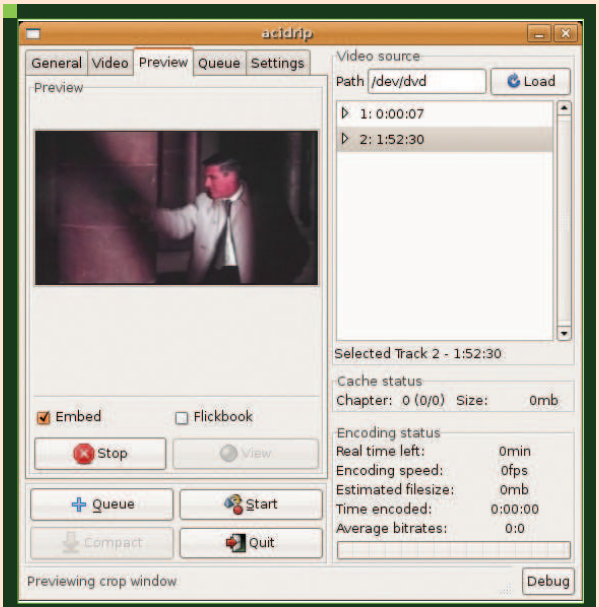
Keressük meg a leghosszabb sávot. Ez tartalmazza magát a filmet. Ha olyan ***DVD***-t archiválunk, amin több extra tartalom is van, akkor számos rövidebb sávot is látunk majd a „lényeg” mellett. Arra sajnos semmi nem utal, hogy a sávok közül melyik tartalmazza magát a lementeni kívánt filmet, tehát rossz esetben előfordulhat, hogy néhányszor próbálkoznunk kell, mire rájövünk, melyik a nyerő. Ha meglel-

tük, akkor az átkódolni kívánt sávot úgy jelölhetjük ki, hogy egyszer rákattintunk.
Ha ezzel megvagyunk, tehát a program már tudja, hogy milyen bemenettel kell dolgoznia, akkor a következő lépés néhány paraméter beállítása. Először is ***General*** (Általános) fülön adjuk meg a sáv címét. Ez lesz majd annak az ***.avi*** kiterjesztésű fájlnak a neve, amiben az eredmény keletkezik. A ***Filename*** mezőben adjuk meg annak a helynek a teljes elérési útvonalát, ahol a fájlt el szeretnénk helyezni. Ennek az információnak ***%T***-re kell végződnie, az alapértelmezett mentési útvonal pedig a ***home*** könyvtárunk. Természetesen bármely olyan helyet megadhatunk célként, ahova van írási jogunk. A fájl méretére és a fájlok számát tartalmazó dobozra később még visszatérünk.
Ha akarunk, megadhatunk néhány metaadatot a lementett filmről az ***Info*** feliratú mezőben. Ide beírhatjuk például a film címét, a főszereplő nevét, a témát, a műfajt, illetve a szerzői jogokkal kapcsolatos információkat. Ezeket az adatokat az ***MPlayer*** képes kiolvasni a visszajátszás során, de amúgy a megadásuk egyáltalán nem fontos és nem is kötelező.
Az ***Audio*** szakaszban a kijelölt nyelvet hagyhatjuk a ***„<Default>English”*** értéken, vagy megadhatunk valamilyen más hangsávot is a legördülő menüből válogatva. Persze ezen a pontosan nem árt az óvatosság, hiszen egyes ***DVD***-ken bizonyos audiósávok csak narrációt tartalmaznak, illetve lehetnek akár teljesen üresek is.
Az ***Audio Codec*** nevű legördülő menüből válasszuk ki, milyen módon szeretnénk a hangadatokat kódolni. A választási lehetőségek természetesen attól is függnek, milyen kodekeket telepítettünk fel a rendszerünkre. Az én gépemen a repertoár a következőképpen fest.: ***copy***, ***pcm***, ***mp3lame***, ***lavc*** és ***faac***. Ami a sebességet illeti, a leggyorsabb az egyszerű másolás (***copy***), mivel ez – nevének megfelelően – csupán átmásolja a hangadatokat a lemezről az archivált anyagba, vagyis egyenesen a keletkező ***AVI*** fájlba. Ha az ***MP3*** kódolás mellett döntünk, amit az ***mp3lame***, vagy a ***lavc*** kodekek segítségével tehetünk meg, akkor lehetőségünk van a bitsebesség megadására, de figyeljünk arra, hogy minél nagyobb sebességet állítunk be, annál tovább tart a feldolgozás. Ha a kódolt anyag túl lassú, vagy túl halk, akkor igény szerint állíthatunk az erősítés mértékén is. Ami engem illet, gyakorlatilag még nem találkoztam olyan anyaggal, ahol erre szükség lett volna, de másoknak esetleg nem lesz ekkora szerencséje.

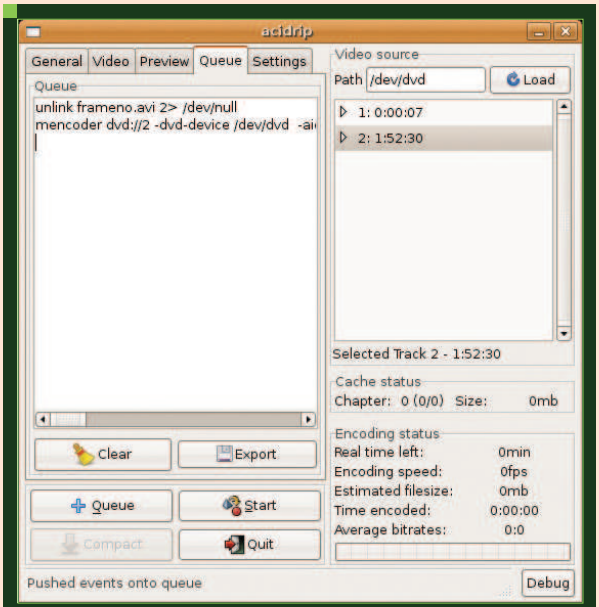


2. ábra Az AcidRip program Video Setting füle

A következő lépés a videó paraméterek beállítása. Kattintsuk tehát a ***Video*** feliratú fülre a megjelenítésükhöz.
A lehetőségek ismét csak attól függnek, hogy milyen kodekek vannak föltelepítve a gépünkre. Az én gépemen a program a következő lehetőségeket kínálta föl: ***copy***, ***raw***, ***nuv***, ***lavc***, ***vfw***, ***qtvideo***, ***libdv***, ***xvid*** és ***x264***. Ha a legkisebb méretben szeretnénk a lehető legjobb minőséget elérni, válasszuk az ***x264*** nevű kodeket. Ami azt illeti, nekem a ***lavc*** kodekkel is egészen kiváló tapasztalataim vannak úgy, hogy minden beállítását az alapértelmezett értéken hagytam. A ***Passes***, ***Bitrate*** és ***Bits/Px*** feliratú mezők tartalmára később még visszatérünk.
A ***crop*** feliratú jelölőmező bejelölése mindig jó ötlet, különösen azonban akkor vehetjük használt, ha széles képernyőre tervezett ***DVD***-t kódolunk át. Nyilván a legkevésbé sem szeretnénk értékes processzoridőt és még értékesebb lemezterületet pocsékolni arra, hogy a kép alján és tetején látható két keskeny fekete csíkot átkódoljuk és tároljuk. Nos, ezzel az opcióval éppen ezt lehet elkerülni. Ha megnyomjuk a ***Detect*** gombot, akkor az ***AcidRip*** az ***MPlayer*** meghívásával bizonyos egyedi képkockák alapján megpróbálja „önállóan” kitalálni a helyes vágási értékeket. Természetesen mi magunk is eljátszhatunk kicsit a ***Width***, ***Height***, ***Horiz*** és ***Vert*** értékekkel, miután azokat nagyjából beállítottuk az automatikus felismeréssel, de én ezzel nem szoktam az időmet tölteni. A tapasztalat azt mutatja, hogy a program elég okos ahhoz, hogy ráhagyjuk, amit magától kitalált. Ha valaki mégis kézzel szeretné a vágást beállítani, akkor olvassa el az erről szóló keretes részt.
Ha egy teljes filmet szeretnénk egyetlen ***CD-ROM***-on tárolni, akkor valószínűleg szükség lesz a képkockák kicsinyítésére is. A műveleti sorrend ebben az esetben úgy néz ki, hogy a ki-



■ *3. ábra* Itt nézhetjük meg a beállítások eredményét. Ha nem látunk képet, vagy nem hallatszik a hang, akkor valószínuleg finomítanunk kell a paramétereken.



■ *4. ábra* A Queue fül. Itt azt a MEncoder parancssort láthatjuk, amit az AcodRip ki fog adni a DVD-n található adatok feldolgozásához.

csinyítés csak a vágás után történik meg, vagyis ne próbálkozzunk olyasmivel, hogy a képből levágunk egy keveset, ha azt látjuk, hogy éppen nem fér el a lemezen a végeredmény, az ott megadott értékek ugyanis nem arra vonatkoznak. Egyszóval adjuk meg a kicsinyítés arányát, és kész. Ha ez nem működik, akkor ezen kell módosítani, más lehetőség nincs. Szintén legyen mindig bejelölve a ***Lock*** aspect nevű jelölőmező,ellenkező esetben torzulhat a kép.

Az utolsó dolog, amit a ***Video*** fülön beállíthatunk az elő- és utószűrő (Pre- és Post filter). Ezeket én rendszerint békén hagyom.

Ha a videótartalom kódolására a ***lavc*** kodeket választjuk, megadhatjuk a bitsebességet (***Bitrate***), illetve a ***Bits/Px*** értéket. Szintén beállíthatjuk a kódolási lépések számát. Általánosságban elmondható, hogy a ***Bits/Px*** érték optimuma ***MPEG-4*** videók esetében valahol 0,249 körül van. Ha kivesszük a pipát a ***Lock*** jelölőmezőből, akkor a bitsebességet (***Bitrate***) kézzel is beállíthatjuk. Változtassuk tehát ezt az értéket addig, amíg a hozzá kapcsolódó ***Bits/Px*** meg nem közelíti az optimumot. A többlépéses kódolás megnövelheti, és általában valóban jelentősen meg is növeli a feldolgozás időtartamát, ugyanakkor határozottan jót tesz a film minőségének – már ami a digitális technikát illeti.

Ha rögzítjük a bitsebességet, akkor semmilyen módon nem tudjuk befolyásolni a keletkező fájl méretét. Miután megadtuk az összes, a képtartalomra vonatkozó beállítást, váltsunk vissza a ***General*** fülre, ahol megnézhetjük, hogy az adott paraméterekkel a program mekkorának becsüli a fájlméretet. A kérdéses mező most már ki lesz töltve. Ha az elérni kívánt fájlméretet kézzel szeretnénk megadni, akkor először vegyük ki a jelet a ***Video*** fülön a ***Lock*** jelölőmezőből (felszabadítva ezzel a bitsebesség zárolását), majd váltsunk vissza a ***General*** fülre, és egyszerűen írjuk be a fájlméretet. Erre általában csak akkor van szükség, ha egy ***DVD*** tartalmát egyetlen, vagy néhány ***CD-R*** lemezre szeretnénk átvinni.

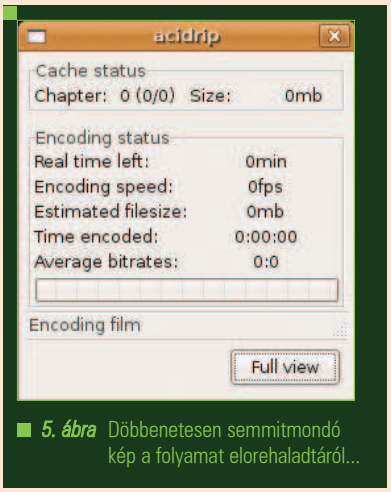
Ha azt tervezzük, hogy a ***DVD*** anyagát ***CD***-kre írjuk, akkor erre több különböző lehetőségünk is van. Az első, hogy az anyagot teljes méretben kódoljuk át, de a programnak megadjuk, hogy több fájlt szeretnénk létrehozni. Négy darab 700 MB méretű ***CD*** általában elegendő egy ***DVD*** átkódolt tartalmának befogadására úgy, hogy a minőség sem romlik érzékelhetően.

A másik módszer szerint először a célfájl méretét adjuk meg, majd a kicsinyítés mértékét úgy állítjuk be hogy, hogy a Bits/Px érték is megfelelő legyen.

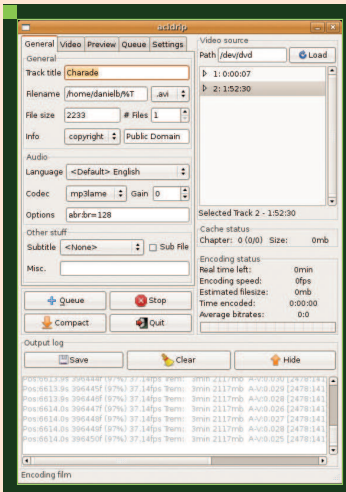
Az átméretezés beállításához használjuk a felfelé és lefelé mutató nyíl billentyűket. Ez azért jó, mert a beállításoknak megfelelő bitsebességet így valós időben láthatjuk a megfelelő mezőben. Sajnos van egy apró problémája a programnak, ami miatt a többi mezőt nem tudja így röptében állítani, maga a módszer azonban minden más, a felhasználó által megadható mezőre működik.

Ha úgy gondoljuk, hogy elértük, amit szerettünk volna, akkor váltsunk át a ***Preview*** (előnézet) fülre. Az ***Embed*** feliratú jelölőmező itt mindig legyen bejelölve, a ***Flipbox*** nevű viszont ne. Ha ezt ellenőriztük, kattintsunk a ***Preview*** gombra. Ha nincs hiba a beállításokban, akkor elindul a film lejátszása. Viszonylag gyakran előfordul, hogy finomítanunk kell a vágási értékeken. Én a beállítások megfelelőségét általában úgy szoktam ellenőrizni, hogy a ***Video Source*** fülön kinézek egy fejezetet a film közepéről, és ebbe kukkantok bele ahelyett, hogy az elejéről indítanám a lejátszást. Aki követi a példámat, az ne felejtse el visszaállí-

tani a mutatót a film elejére, mielőtt elkezdené a tényleges átkódolást.
Ha eleget láttunk ahhoz, hogy biztosak lehessünk a beállítások helyességében, nyomjuk meg a ***Stop*** gombot.
Ha elégedettek vagyunk a beállításokkal, akkor észen is állunk arra, hogy a filmet a feldolgozási sorba küldjük a ***Queue*** gomb megnyomásával.
Ha átváltunk a ***Queue*** fülre, akkor két lehetőség közül választhatunk: törölhetjük az aktuális feldolgozási listát, vagy exportálhatjuk azt egy héjprogram formájában. A soros feldolgozás természetesen azt jelenti, hogy akár több feladatot is sorba állíthatunk, ami akkor különösen hasznos, ha több kisebb, a lemezen extraként szereplő szakaszt akarunk átkódolni külön fájlokba.
Ha elkészültünk a feldolgozási sor összeállításával, benne van minden, amit át szeretnénk kódolni, akkor a kezdéshez nyomjuk meg a Start gombot. Erre megjelenik egy kis ablak, amelyben elvileg a folyamat előrehaladását kísérhetnénk nyomon, de amelyben ténylegesen nem látszik semmi. Ez valószínűleg az ***AcodRip*** leglátványosabb hibája, már amennyiben ezt a képet látványosnak lehet nevezni. Persze az is lehet, hogy mire ez a cikk a nyomdába kerül, már ki is javították a fejlesztők, de tény, hogy nekem a 0.14-es változat (A ***MEncoder 1.0pre8*** változatával kombinálva) még így működött. Persze a ***MEncoder*** maga ettől még tökéletesen működik, csak fogalmunk sincs, hol tarthat.
Ha mégis látni szeretnénk a folyamat előrehaladtából valamit, kattintsunk a ***Full view*** feliratú gombra, amivel visszatérhetünk az eredeti felhasználói felülethez. Itt a ***Debug*** gombra kattintva láthatjuk a ***MEncoder*** nyers kimenetét. Görgessük a képet a lista aljára, ahol a kódolás előrehaladása végül is követhető. Ebben az ablakban természetesen azt is láthatjuk, ha az ***AcidRip*** valamiért nem tudta a kijelölt feladatot elvégezni. Ilyenkor az esetek többségében szerencsére kapunk néhány olyan üzenetet, amelyekből kideríthető, hogy hol, melyik beállítással lehetett a gond.
Az utolsó olyan ül, aminek a tartalmáról még nem esett szó a ***Settings*** (beállítások). Itt a legkülönbözőbb dolgokat lehet egy kicsit eljátszani. Megadhatjuk például, hogy milyen útvonalon érhető el a ***MEncoder*** és az ***MPlayer***. Erre amúgy csak akkor lesz szükségünk, ha valamilyen nem szokványos helyre telepítettük a két alkalmazást. Beállíthatunk aztán olyan dolgokat is, amelyek egyáltalán nem ennyire maguktól értetődőek.
Összességében az ***AcidRip*** egy nagyon hasznos kis alkalmazás, legalábbis nekem nagyon sokat segített abban, hogy megmenthessem a ***DVD***-im tartalmát. Igaz ugyan, hogy a felhasználói felületén van még itt-ott mit csiszolni, de ezektől az apróbb „felületi egyenetlenségektől" eltekintve már most is kiválóan működik.
Az egyetlen rossz hír az ***AcidRippel*** kapcsolatban az, hogy a szerzője, ***Chris Pillips*** saját elmondása szerint nem sok időt szeretne a jövőben a program frissítésével tölteni, sőt az sem kizárt, hogy semennyit.
A nyílt forrásnak persze megvan az a szépsége, hogy egy program fejlődését egy ilyen esemény nem föltétlen kell hogy meggátolja. A fejlesztést tulajdonképpen bárki átveheti, aki késztetést és kellő erőt érez magában, no és persze aki annyira ért a ***Perl***-hez, mint az eredeti szerző.
Akadnak esetleg máris jelentkezők?



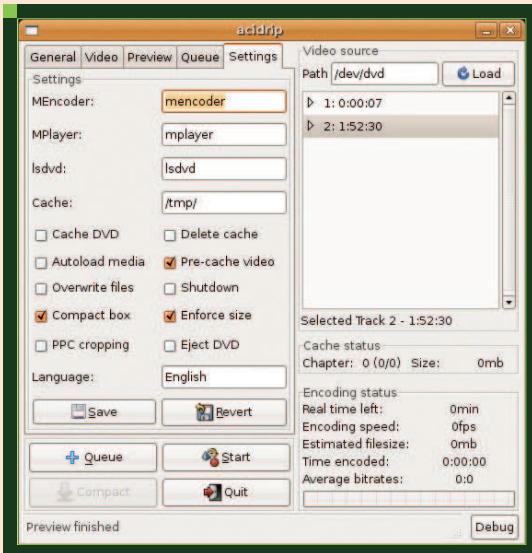
■ ***5. ábra*** Döbbenetesen semmitmondó kép a folyamat elorehaladtáról...



■ ***6. ábra*** A döbbenetesen informatív kimeneti napló

Különleges kódolási beállítások

Most, hogy az ***AcidRip*** segítségével már egészen jól mennek a dolgok, elhatároztam, hogy a következő lépésben magát a ***MEncodert*** fogom használni, mindenféle mankó nélkül.

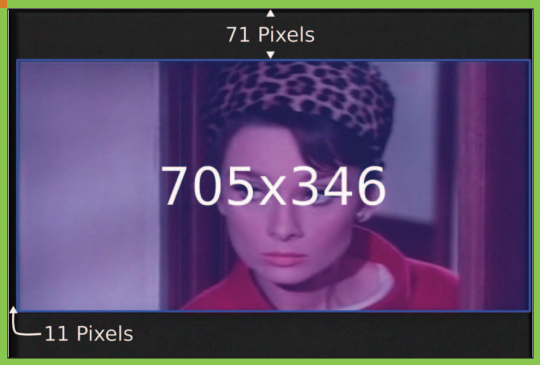


■ ***7. ábra*** Az AcidRip beállításokat tartalmazó füle

Kézi vágás

Egyes ***DVD***-k tartalmával az ***AcidRipnek*** meggyűlhet a baja a vágás során, mivel a kép széle nem kellően egyenes. Ilyenkor a ***„crop failed”*** üzenetet kapjuk, amit ezek után csak a vágási határok kézi beállításával lehet orvosolni. Ha beleszaladtunk ebbe a problémába, akkor a beállítást mindenképpen 720 pixel széles vágással kezdjük, a magasság pedig legyen 480 pixel. Innen induljunk el aztán lassan lefelé. A kezdő beállítást az indokolja, hogy a 720x480 pixel a szabványos ***NTSC DVD*** képkocka mérete. ***PAL*** filmek esetében a helyes kezdő beállítás a 720x568 pixel.
Kézi vágásnál a ***Horiz*** és ***Vert*** értékek beállítása elsőre meglehetősen zavaros lehet. Ez a két szám a kép vízszintes és függőleges irányú eltolását jelenti a bal felső saroktól, vagyis attól a ponttól kezdve, ahol a teljes méretű képkocka kezdődne. A kivágási keretet természetesen az imént említett szélesség és magasságadat határozza meg.

I. ábra Képkocka a Charad című filmből



II. ábra A fenti filmmel kapcsolatban legjobbnak talált vágási beállítások

Vegyük például az ***I. ábrán*** látható, amúgy a ***Charad*** című filmből származó képkockát. (A főszerepekben ***Audrey Hepburn*** és ***Cary Grant***.) Ez a film két okból is kiválóan alkalmas példának. Először is az ***AcidRip*** képtelen nála automatikusan meghatározni a vágási paramétereket. Másodszor egy az ***Egyesült Államok*** szerzői jogokról szóló törvényének megfogalmazásában vétett hiba miatt ez a film kibocsátásakor a szabadon felhasználható kategóriába esett, ezért ma bárki, bármilyen célra felhasználhatja, még arra is, amire én most.

Amint az a *I. ábrán* látható, a filmkocka bal oldalán van egy keskeny fekete csík, ami mellesleg az egész filmre igaz. Ugyanilyen kis üres terület van a jobb oldalon is. Ez eddig rendben is volna, hiszen levághatjuk őket. Csakhogy mindkét fekete terület zajos, ami az automatikus detektálást igen megnehezíti. A felső és alsó szegélyek szintén „mocorognak” egy kicsit a film során. Mindezeket figyelembe véve én a következő kézi vágási szintek beállítását találtam a legkielégítőbbnek:

Width: 705
Height: 346
Horiz: 11
Vert: 71

A kivágási téglalap szélessége tehát 705 pixel, míg a magassága 346 képpont. Ezt a négyszöget a bal felső saroktól mérve balra 11 pixellel, lefelé 71 pixellel kell eltolni ahhoz, hogy a kép hasznos részének lehető legnagyobb hányada essen bele.
A fenti beállításokhoz úgy jutottam el, hogy kezdőértéknek 700 pixel szélességet és 400 pixel magasságot adtam meg, majd ezeket fokozatosan változtattam, és figyeltem a hatást. A vízszintes eltolás kezdetben 10 pixel, míg a függőleges 60 pixel volt. Szemre ezeket véltem a legjobbnak. Ezek megadása után oda-vissza kapcsolgattam a ***Video*** és a ***Preview*** fülek között, és finomra hangoltam a kezdeti beállításokat.

Ebben persze kezdetben az ***AcidRip*** kimenete is segítségemre lesz majd, hiszen a program arra is képes, hogy a beállított paramétereknek megfelelő utasításokat egy héjprogram formájában lemezre mentse. Ezek olyan parancsok, amelyeket mi magunk is átadhatunk a ***MEncodernek***, így jó alapul szolgálnak a „kézi vezérléshez”. Természetesen a ***MEncoder*** dokumentációja is kiváló tudásforrás, hiszen megtaláljuk benne az összes olyan beállítható paraméter leírását, amelyekkel a legjobb képminőséget érhetjük el. Most már csak a kellő mennyiségű szabadidőt kell valahogy megteremtenem.

Linux Journal 2006., 152. szám

A cikk forrásai:
➲ ***www.linuxjournal.com/article/9389***

Daniel Bartholomew a korai 1980-as évek óta használ számítógépeket, ez volt ugyanis az az időpont, amikor a szüleitől kapott egy Apple IIe-t. Aztán 1996-ban, amikor már szűkösnek érezte mind a Mac-et, mind a Windowst, felfedezte a Linuxot. Azóta számos különféle terjesztést használt sikerrel. Jelenleg Észak Karolinában él feleségével és gyermekeivel.

Open Arena – Quake3 után, szabadon

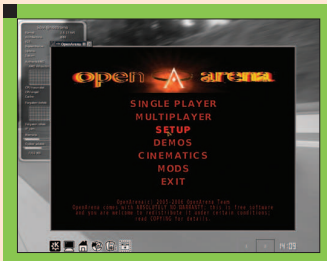
Az id Software fenegyerekeire gondolva valószínűleg senki sem kérdőjelezi meg az FPS kategória dobogós helyezéseit... Amondó vagyok, a Quake sorozat harmadik tagját próbáljuk meg úgy életre hívni, hogy közben szem előtt tartjuk a Linux rendszereket övező szabadságot!

1999... Ugye, milyen régen írtuk le ezt a dátumot? Ha „öreg kvékerként" emlékezem vissza ezekre az időkre, önkéntelenül beugrik *John Carmack* csapatának formabontó projektje: a *Quake* vonalat ebben az évben helyezték át *Multiplayer* alapokra, és készítettek hozzá egy olyan grafikai / fizikai motort, amely még a mai túlhaladott állapotában sem kelt szégyenérzetet senkiben. Bizony, hét év telt el a publikálás óta, ennek ellenére 2007-ben még rendületlenül jelennek meg újabb és újabb életképes *Arena* modifikációk. 2006-ban nyílt meg a *Quake* harmadik részének motorja, *GPL* licenc szerint. Az előző epizódok alapkódja ennél jóval korábban „szabadult", így kicsit meglepően hangzik a hat-hét év „bezártság". Ennek magyarázata rém egyszerű: a csapat egy esztendővel korábban még mindig mérhető anyagi hasznot tudott kipréselni a játékból. Sebaj, ami késik, nem múlik: a *C* programsorok tavalyi publikálását követően profi programozók szinte azonnal „megpatkolták" a forráskódot, majd közkinccsé tették a több rendszerre elkészített friss binárisokat. Ugye, nem okozok nagy meglepetést, ha a szakértőket említve az *Icculus.org* csapatát nevezem meg?

A körítés

Ezen a ponton talán sokakban felmerülhet a kérdés: „Ugyan már, mit lehetett változtatni, finomítani a gyári binárison?". Igazából arról van szó, hogy *John*-ék a *v1.32b* verziónál megálltak, és nem foglalkoztak tovább a *Quake3 Point Release* kiadásával. Ez a változat a 32 bites *Linux* rendszerek java részén jól fut (habár mára már jellemzően hang és *DGA* problémával szembesíti a felhasználókat), azonban a 64 bites (vagy éppen az *SMP*) rendszereken nem hívható elő a hardver teljes kihasználtságát szem előtt tartva. Elkészült tehát az *Icculus* naprakész változata, mely ráadásul *Windows*, *Linux*, *Solaris*, *OS X* felületen egyaránt kompatibilis az eredeti kóddal és egymással is.

Az *id Software* adatcsomagjait azonban továbbra is licencek kötötték „karóhoz". Ez a jelenség két ponton is sántított: a mezei felhasználók még mindig cirka 6-*7000 Forintnak* megfelelő összeget kellett fizessenek a teljes játékért (ami alapvetően azért mégis csak túlhaladott), másrészről a *MOD*/*Fork* fejlesztők nem használhatták fel az eredeti textúrákat és modelleket munkáikban. Ez utóbbi jelenség egy új játék (vagy egy színvonalas modifikáció) megalkotását akár egy egész évvel is hátráltathatta...



1. ábra Íme az Open Arena, KDE asztalon

2. ábra Quake alapú terep-implement

3. ábra A látvány sokszor idézi az eredetit

4. ábra Nincs ok a panaszra, a fizika ismerős...

Sőt, ilyenkor talán még az is bonyolítja a helyzetet, hogy az adatokat sajnos nem is lehet pontosan lemásolni (jogi okok miatt). Azonban az imitáción túllépve néhány fejlesztő egy új, Szabad *Quake3*-at álmodott az asztalra. Úgy néz ki, komolyan gondolták, hiszen a ➲ http://openarena.ws oldalon lázas munka folyik: készül az *Open Arena*! A cikk írásakor elérhető legfrissebb kiadás még a kezdetleges állapotot sejtető *v0.6.0* azonosítót viseli, ennek ellenére biztonsággal használható. Hat pálya, négy modell és a teljes funkcionalitás már egy kisebb házi bajnokságot is vígan elbír.

Hozzávalók, több személyre

Adott tehát az *Icculus* által finomított alapkód, erre építkezik az *Open Arena*. Szerencsére az egész projekt egy-két húzásból felélesztheto, hiszen a linuxos *tarball* minden szükséges elemet tartalmaz. Az előbb írt (hivatalos) oldalra látogatva töltsük le a cirka 80 *MByte* méretű archívot. Csomagoljuk ki egy mindenki által olvasható területre (például a */usr/local/games* könyvtárba), majd vegyük szemügyre a létrejött mappát! Az *id Software* munkáira jellemző, ismerős szerkezeten túl hat *ELF* bináris található itt: ezekkel indítható *i386*, *x86/64*, *SMP* (több processzoros) környezetben a játék, kliens és dedikált szerver üzemmódban. A ránk vonatkozó állományt linkeljük ki egy elérési útra, majd adjunk rá futtatási jogot mindenki részére. Esetemben ez a művelet így néz ki:

```
ln -s /usr/local/games/
➥oarena/ioquake3.i386 /usr/
➥local/bin/oarena
chmod a+x /usr/local/bin/oarena
```

Ezzel a linkkel, pontosabban a felhasználóként kiadott paranccsal máris indítható a játék. A program minden beállítását és mentését a játékos személyes mappájában tárolja, a */home/$/.openarena/baseoa* rejtett úton. Hardverigény tekintetében nincs számottevő eltérés az eredeti *Quake3*-hoz képest. Az *Open Arena* 1 *GHz* központi egységgel, 192 *MByte* memóriával és egy második generációs *3D* gyorsítóval (működő *DRI* illetve *GLX* kapoccsal) már nagyobb felbontásban is vígan fut. Ezeket a paramétereket a négy-öt évvel ezelőtt vásárolt *PC*-k is teljesítik, így ha valaki érdeklődik a szoftver iránt, valószínűleg nem fog problémába ütközni a használatát illetően.

Ami elvárható

A jelenlegi verzióban az egyéni és csapatmódok mellett természetesen már működik a zászlólopás is. Még a „kommand" lehetőség is elérhető, tehát a *BOT*-oknak parancsok oszthatóak (akár bázis védelmet, vagy éppen önálló tevékenységet kérve). Fontos kérdés lehet a minőség: ez amolyan „jó erős középszerű" – a modellek és a pályák némelyike a *Quake* első részéből költözött át, de azért akad olyan terep is, mely minden tekintetben az eredeti *Arena*-t idézi (a mellékelt képek reményeim szerint árulkodnak erről). A program összességében figyelemre méltó: ha a modellek kivitelezése még tovább javul, akkor semmi akadálya sem lesz a szélesebb (el)ismertségnek. A pályák és kasztok számának növekedése szintén a jövő zenéje: úgy gondolom, megéri „tűkön ülve" várni a soron következő változásokat...

Kovács Zsolt (kovi@linuxforum.hu)
Quake fanatikus. Négy éve a debreceni linuxosok egyike. Töretlenül hisz a Slackware terjesztésben.