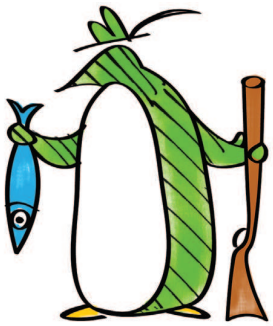
© Kiskapu Kft. Minden jog fenntartva

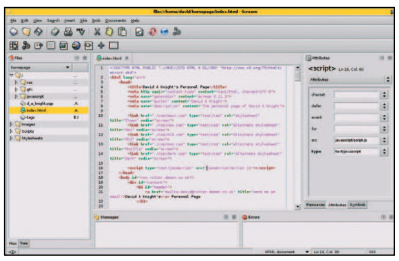
Programvadászat



A korongunk legnagyobb részét a Munjoy Linux foglalja el, amely egy Debian és Knoppix alapú terjesztés, automatikus hardverfelismeréssel, a legfrissebb programokkal, és könnyen használható telepítővel. Mivel Debian alapokon nyugszik nem kell aggódni a programok hiánya miatt. Nyugodtan hozzáadhatjuk a nekünk tetsző sorokat a ***sources.list*** fájlhoz és máris rendelkezésünkre áll a teljes Debian archívum. A fórumokon többen is megkérdezték, nem jelenthet-e gondot az, hogy a SID, vagyis az Unstable ágat használják a fejlesztők? Nos, ahhoz, hogy egy Debian alapú rendszer szinte naprakész legyen ez az egyedüli megoldás. Így viszont mindig a legfrissebb programokat kapjuk kézhez, nem kell két évet várni a következő megbízható Debian megjelenéséig. Az abban lévő csomagok amúgy akkor már régen elavultak ahhoz, hogy asztali munkakörnyezetben használjuk őket.
Az automatikus hardverfelismerésnek köszönhetően nem jelenthet gondot a számítógép alkatrészeinek cseréje, valami megváltozik a rendszerünkben, azt önműködően felismeri és betölti hozzá a megfelelő rendszermag modulokat. Az X beállítása a Knoppixnál megszokott módon a `mkxf86config` programmal történik.
A fejlesztők több erőforrásigényes alkalmazást i686-ra fordítanak, amivel jelentős sebességnövekedés érnek el.

SCREEM

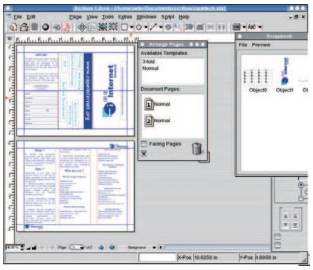
A SCREEM (Site Creation and Editing EnvironMent) segítségével gyorsan és egyszerűen készíthetünk és tarthatunk karban weboldalakat vagy egész siteokat. Az anyag feltöltésekor a következő protokollok valamelyikével kapcsolódhatunk a kiszolgálóhoz: FTP, WebDAV, SCP/RCP és CVS. A SCREEM nem WYSIWYG, azaz nem „amit látsz



azt kapod” HTML szerkesztő, hanem egy kifinomult felülettel rendelkező szövegszerkesztő. A vele elkészített kód a grafikus szerkesztőkkel ellentétben, amelyek hajlamosak „teleszemetelni” a kódot mindenféle haszontalan dologgal tiszta HTML. Akik egy jó és sokoldalú HTML szerkesztőre vágynak azoknak feltétlenül ajánlható.

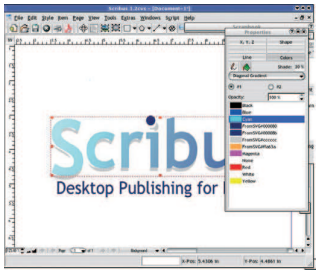
Scribus

A Scribus neve valószínűleg ismerős azoknak akik foglalkoznak DTP-vel és Linuxszal. Augusztus végén közel egyéves fejlesztői munka után megjelent a Scribus 1.2 „aKademy Edition” változat. Az 1.0-ás változathoz képes nagyon sok változás és természetesen minőségi javulás is tapasztalható.

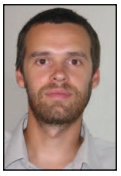


Nagy vonalakban a főbb változások:

- Új EPS/PS importszűrő, az EPS és PS fájlok vektoros importálását teszi lehetővé, amely után normál scribus objektumként tudunk velük dolgozni.
- Több mint 800 hibajavítás és felhasználói kérés került javításra/megvalósításra.
- A felhasználói felületet 27 nyelvre lefordították.
- A Scribus támogatja a ***New from/ Save as Template*** bővítményt, ennek segítségével a profik és a kezdők is gyorsan elkészíthetnek előre formázott dokumentumokat.
- A ➲ http://docs.scribus.net oldalon elérhető a teljes dokumentáció.



A Scribus fejlesztői szeretnék, hogy minden terjesztés minél előbb ezt a változatot használja. A fejlesztők vezetője ***Franz Schmid***, a többi tag profi programozókból, DTP szakemberekből áll, a közös céljuk a Scribus felhasználóbarát de magas szintű DTP-s programmá fejlesztése.
A jelenlegi változattal a Scribus felnőtt korba lépett, mivel egy kereskedelmi újságkiadó, a Twin Tiers Times (egy USA hetilap) az elmúlt hónapban már ezt a nyíltforrású programot használta a nyomdai előkészítésre, olyan programok társaságában mint a GIMP 2.0 és az Inkscape.



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

A doboz nem kell

Új, *G5*-ös processzorra épülő *iMac* gépet mutatott be a *Macintosh*. A cég hagyományaihoz híven az egész gép – eltekintve persze a billentyűzettől és az egértől – egyetlen dobozba került, ám a doboznak felénk néző része a kor színvonalának megfelelően immár nem CRT monitor, hanem 17" vagy 20" átmérőjű, szélesvásznú TFT.

A mindössze 5 cm vastag, hajlított konzoljával egészen vegytiszta hatást keltő munkaállomások kialakítására is alkalmas gép minden földi jóval, többek közt vezeték nélküli vagy *Bluetooth*-csatolóval is felszerelhető, a különféle egységeket pedig USB kapukon keresztül csatlakoztathatjuk hozzá.

Maroknyi Linux

A *Wildseed* termékeként megjelent az első *Linux* alapú, GSM/GPRS hálózatokra szánt mobiltelefon az amerikai piacon. A szokatlan, ívelt formájú készülék érdekessége, hogy fedőlapjának, pontosabban „bőrének" cseréjével egészen új személyiséggel ruházható fel. Az intelligens bőrök – ilyenekből máris 23-féle érhető el – egy apró lapkát rejtenek magukban, amely soros összeköttetésen keresztül kapcsolatot tart a készülékkel. A lapka csengőhangokat, egyéb hangjelzéseket, animációkat és egyebeket tárol, de külön gombokkal akár arra is alkalmas lehet, hogy játékkonzollá változtassa a telefont.

A készülék központi részébe 200 MHz órajelű *XScale* processzor került, rajta 2.4.5-ös *Linux* fut. A GSM hálózattal való kapcsolattartásról egy különálló kommunikációs processzor gondoskodik. Ezen egy önálló, valós idejű operációs rendszer fut. Az okos bőrökben szintén egyedi operációs rendszer fut egy 8 bites *RISC* processzoron.
➲ www.smartskins.com

Az ablakos gombot lehagyták

A hazánkban is ismert német *Cherry* a *SuSE* támogatásával linuxos billentyűzet gyártását tervezi. A billentyűzet

a *CyMotion* sorozat egy átalakított példánya lesz, amelyre különleges, a cég által mellékelt programmal testreszabható szolgáltatást nyújtó gombok kerülnek – köztük egy *Tux* rajzzal is ellátott. A billentyűzet ára meglehetősen borsos lesz, alulról súrolja a tízezer forintos határt, ráadásul egyelőre csak *Németországban*, *Hollandiában* és *Angliában* tervezik forgalmazni, tehát magyar gombokkal ellátott változatra nem számíthatunk.
➲ www.cherrycorp.com

Vége lehet a szemétáradatnak

A *Sendmail* fejlesztői elkészültek a kéretlen levelek szűrésére szánt moduljuk első változatával. A modul szabadon letölthető a *Sendmail* oldaláról.

Elérhetővé tételének egyelőre hangsúlyozottan az a célja, hogy minél szélesebb körben terjedjen a megoldás, illetve valós környezetekben tudjanak tapasztalatokat gyűjteni a működésével kapcsolatban – ugyanakkor kulcsfontosságú rendszereken nem javasolják a használatát.
A *Sender ID* nevű, egyelőre csak leendő *IETF* szabvány egy részét megvalósító modul a levelek fogadásakor megpróbálkozik a küldő hitelesítésével, így előzve meg a szemét eljutását a felhasználói postafiókokig.
A *Sender ID*, bár alig jelent meg, máris viták tárgyává vált, ugyanis a Microsoft által fejlesztett *Caller ID for e-mail* és a ➲ Pobox.com-ot alapító *Meng Wong* által kidolgozott *Sender Policy Framework* (lásd: *SPF* bemutató, *Linuxvilág*, 2004. augusztusi szám) keresztezéséből jött létre, és a Microsoft által támasztott felhasználási feltételek nem mindenkinek nyerték el a tetszését.
➲ www.sendmail.net

Fényképező kesztyű

A felölthető számítógép mintájára a *Fuji* előállt a felölthető fényképezőgép ötletével. Az apró fényképezőgép leginkább valamiféle kesztyűre hasonlítana, a lencse egy apró gyűrű formájában a középső ujjra kerülne, az LCD-kijelző a tenyér felőli oldalon kapna helyet, míg az áramellátást biztosító akkumulátort valahova a csukló környékére helyeznék. A cég elképzelése szerint a gépet a viselő személy izmainak mozgatásakor keltett elektromos jelek hoznák működésbe, például amikor a tulaj V betűt formál középső és mutatóujjával. A meglévő műszaki megoldások már lehetővé teszik egy ilyen termék tömeggyártását, most már csak a megfelelő alkalmazási területet kell megtalálni.

Mindent látnak

CameraProbe8 névvel új adatközpont-felügyelő készüléket dobott piacra a bangkoki *AKCP* nevű cég.
A *CameraProbe8* egyrészt rendelkezik egy beépített kamerával, amely állóés mozgóképeket egyaránt tud adni a megfigyelt területről, másrészt a legkülönfélébb – ajtónyitás-, mozgás-, víz-, légmozgás-, füst-, hőmérséklet- – érzékelőket lehet hozzá csatlakoztatni, így segítségével egy-egy helyiség vagy terület állapotát gyakorlatilag minden szempontból figyelemmel lehet kísérni. A *CameraProbe8* a *Nagios* hálózat-felügyeleti alkalmazással együttműködve nemcsak fizikai, de szoftveres eseményeket, állapotokat is képes követni, a kapott eredményeket pedig többféle formátumban, akár grafikonok rajzolásával képes megjeleníteni.

A készülék *SNMP* és *HTTP* protokollon keresztül, biztonságosan felügyelhető.
A *CameraProbe8* egy 32 bites *ARM* processzorra épül, operációs rendszere *Linux*, amely egy 64 MB kapacitású *Flash* memóriából indul. Hálózatra csatlakoztatását egy *Ethernet* kapu teszi lehetővé, ezen felül két darab soros kapuval és nyolc darab érzékelőkapuval rendelkezik. Ára 850 dollártól, vagyis valamivel kevesebb, mint kétszázezer forinttól indul.
➲ www.akcp.com

© Kiskapu Kft. Minden jog fenntartva

© Kiskapu Kft. Minden jog fenntartva

Új kommunikátorok

Hamarosan két új kommunikátorral is bővül a *Nokia* kínálata: a *9500*-as „féltégla" az idei év utolsó negyedében, míg a *9300*-as modell jövő év elején lesz elérhető. A *9500* -as valóban

mindent tud, amit ma egy mobiltelefon - személyi titkár készüléktől elvárni érdemes. A teljesség igénye nélkül: mindhárom GSM sávon használható, belső és külső kijelzője egyaránt színes, teljes értékű billentyűzettel rendelkezik, támogatja az *EDGE*, az *IEEE 802.11b* és a *Bluetooth* kapcsolatokat, a *WPA*, *SSL/TLS*, *VPN* és *IPSec* protokollokat illetve megoldásokat, továbbá tartalmazza a mobil munkavégzéshez szükséges szövegszerkesztőt, böngészőt, táblázatkezelőt és egyéb alkalmazásokat.
A *9300*-as típus kicsivel butább nagyobb testvérénél, például hiányzik belőle a kamera és a vezeték nélküli hálózati csatoló, ám ettől eltekintve hasonló képességekkel rendelkezik. Az elmaradt szolgáltatásokért némi ellentételezés, hogy az alacsonyabb típusjelű telefon könnyebb és némileg kisebb is társánál.
➲ www.nokia.com

Osztódnak a magok

Egyre élesebb a verseny az *Intel* és az *AMD* között, ami egyelőre az egymásra licitálásban mutatkozik meg: ki tud szebbeket mondani eljövendő kétmagos processzorairól? Az *IDF*-en az Intel kétmagos *Itanium* és asztali processzort is bemutatott, de az AMD sem rest, jövőre kétmagos *Opteron* lapkákat ígért vásárlóinak, amelyek támogatását – az ígérgetéseknek némi komolyságot adva – az *IBM* már be is jelentette. Ahogy mondani szokás, ne tartsuk vissza a lélegzetünket, hiszen egyik cégtől sem áll távol a bejelentett termékek tényleges piacra dobásának csúsztatása, valamint meglepő lenne, ha a még el sem terjedt 64 bites lapkák felbukkanása után ismét ilyen komoly teljesítménybeli ugrást követnének el az iparág szereplői.
Mindeközben az AMD az amerikai piacon képes komolyan megszorongatni a hagyományosan piacvezető Intelt: idén nyáron több olyan hét is volt, amikor az eladott asztali gépek nagyobb hányadába került AMD processzor, mint Intel; márpedig ilyenre hosszú évek óta nem volt példa. Egyvalakinek mindenképpen jót hoz a kiélezett versengés, és ez a tisztelt fogyasztó, aki ugyanakkora vételár fejében egyre komolyabb teljesítményű eszközökhöz juthat.

Naptármadár

A *Mozilla.org* kiadta önálló naptár alkalmazásának, a *Sunbirdnek* (napmadárnak) első tesztváltozatát. A *Sunbird* – ez egyébként csak ideiglenes név,

a végleges program más elnevezést fog kapni – tervezet a *Mozilla* naptár összetevőjének újratervezett változata, fejlesztésének alapgondolata egy többféle operációs rendszeren is futtatható időkezelő alkalmazás létrehozása, amely a Mozilla hasonló összetevőjéhez képest kisebb méretű és nagyobb teljesítményű. A tervezet kitűzött céljait áttekintve annyi már most megállapítható, hogy a majdani alkalmazás széles körű együttműködésre lesz képes mind csoportmunka-kiszolgálókkal, mind hordozható eszközökkel és mobiltelefonokkal, továbbá rengeteg finom beállítási lehetőséget kínál majd használóinak.
➲ www.mozilla.org/projects/calendar/

Csábít a Sybase

A *Sybase* bejelentette, hogy *Adaptive Server Enterprise* (*ASE*) adatbázis-kiszolgálójának egy változatát ingyenesen teszi elérhetővé *Linux* alá. A cég képviselője szerint lépésük célja az volt, hogy a költséghatékonyabb megoldásokat kereső, és ezért a *Linuxszal* ismerkedni kívánó felhasználók számára ingyenes tesztelési lehetőséget biztosíthassanak. Az ingyenes *ASE Express Edition for Linux* az *ASE 12.5.2* változat alapvető szolgáltatásait képes biztosítani, ám legfeljebb egy processzoron futtatható, maximálisan 5 GB adatot képes tárolni, és memóriából is csak 2 GB-ot hajlandó használni. A kiszolgáló forráskódját nem adták ki, a *Sybase* szerint a tényleges felhasználók számára ennek nincs is jelentősége, őket inkább a termék érdekli. Az *ASE* szabaddá tett változata 32 és 64 bites gépeken egyaránt használható.

OpenGL 2.0

Az *OpenGL* szabványok sorozatában hetedik tagként megjelent az *OpenGL 2.0*. A 2.x sorozat nyitó tagjának kiadásával az OpenGL Shading Language is az alapcsomag részévé vált, de további kisebb-nagyobb változások, fejlesztések is történtek az előző, 1.5-ös változathoz képest. Az új előírás-gyűjtemény természetesen képes visszafelé együttműködni a korábbi változatokkal. A dokumentumot bárki szabadon letöltheti az *OpenGL* weboldalról.
➲ www.opengl.org

Scribus 1.2

Az 1.0-ás változat kiadása után több mint egy évvel megjelent a *Scribus* 1.2-es változata. A nyílt forrású kiadványszerkesztő programban a felhasználók kérésére megvalósított szolgáltatások és a kijavított hibák száma együttesen eléri a nyolcszázat, és felhasználói felületét is nem kevesebb, mint 27 nyelvre fordították le. A fejlesztő csapat ezzel egy időben büszkén jelentette be, hogy az amerikai *Twin Tier Times* hetilapot *Scribus*, *GIMP 2.0* és egyéb nyílt forrású alkalmazások segítségével szerkesztik, vagyis a kereskedelmi felhasználás is megkezdődött. További újdonság, hogy ➲ docs.scribus.net cím alatt egy a leírásokat összefoglaló weboldal is indult, amelyet idővel szintén többnyelvűre fognak bővíteni.
➲ www.scribus.net

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

Mi újság a rendszermag fejlesztése körül?

ELSA (Enhanced Linux System Accounting) néven elindult egy új, a folyamatok kezelésével kapcsolatos projekt. A folyamatok egyedi kezelésére természetesen már számos jól bevált módszer és eljárás létezik. Amit az *ELSA* kínál, az a folyamatok csoportokba szervezése, és közös kezelése. Ezen kívül elméletileg a rendszer számos egyéb tulajdonságából is képezhetünk majd csoportokat, amely szintén segítheti a működés átfogó elemzését.

Az *InterMezzo* fájlrendszer támogatása kikerült a 2.6-os rendszermagból, mivel pillanatnyilag sem alkotója, *Peter J. Braam,* sem senki más nem kívánja továbbfejleszteni. *Peter* maga is elismerte, hogy a fájlrendszer támogatása minden probléma nélkül megoldható különálló modulként is, így egyetértett *Linus Torvalds* döntésével.

Az *InterMezzot*, melyet szerzője eredetileg nagy méretű klaszterekhez, kiszolgálók tartalmának többszörözéséhez, mobil számításokhoz, és egyéb, nagy rendelkezésre állást igénylő műveletek támogatásához tervezett ténylegesen 2002 óta nem tartotta karban senki. A fejlesztéshez kapcsolódó levelezési listákon is halotti csend uralkodott már jó ideje. Mindezek valószínűtlenné teszik, hogy az *InterMezzo* egyhamar visszatérjen az aktív magforrásba.

Bár a 2.6-os rendszermag egyre stabilabb, a 2.4-es sorozat fejlődése sem állt meg. Ma is számos olyan felhasználó van, aki inkább a régebbi változatot telepíti, amelybe lassan a 2.6-os mag új szolgáltatásai is beépülnek. *Marcello Tosatti*, aki a 2.4-es fát tartja karban, eredetileg lassítani akarta ezt a folyamatot, mivel arra számított, hogy akinek szüksége van az új szolgáltatásokra, az úgyis át fog térni a 2.6-os kernelre. Valójában azonban nem ez történt. Mivel a 2.6-os sorozat még nem stabilizálódott teljesen, *Tosattitól* rengetegen követelték, hogy vegyen be a 2.4-esbe olyan szolgáltatásokat, mint amilyen például a *SATA (Serial ATA)* merevlemezek támogatása. A problémát az okozza, hogy normál körülmények között egy stabil rendszermagba nem szokás felvenni olyan szolgáltatásokat, amelyek destabilizálhatják azt. Mindeközben *Marcello* keményen dolgozott a 2.6-os mag egyes új szolgáltatásain is. 2004 áprilisában ő küldött el *Andrew Mortonnak*, a 2.6-os fa karbantartójának néhány olyan kódrészletet, amelyek lehetővé teszik bizonyos küszöbök felhasználónkénti beállítását. (A 2.6-os változat karbantartásában egyébként *Linus* is részt vesz egészen addig, amíg nem indul útjára a 2.7-es.)

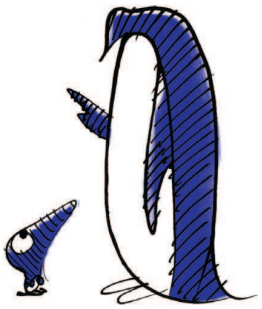
Az említett küszöbök bevezetését egyébként az a felismerés indokolta, hogy rosszindulatú felhasználók szolgálatmegtagadási támadást (DoS) intézhetnek a rendszer ellen olyan alkalmazásokkal, amelyek túlságosan sok feldolgozásra váró jelet állítanak elő. Sajnos megeshet, hogy egyedül a rendszermag módosítása nem is lesz elég a rendszer biztonságának eléréshez. Szükség lehet egyes héjak (bash, csh és egyebek) átírására is. ami sajnos kompatibilitási gondokat okozhat, hiszen a felhasználók nem tehetik meg, hogy csak a rendszermagot cserélik le.

Nem múlik el hónap, hogy valakinek ne támadva valami igazán rendhagyó ötlete, amit aztán ráadásul meg is valósít. Bár egyesek szerint maga a Linux is ilyen ötlet, nemrég felbukkant egy újabb kiváló példa. *Szergej Lozovsky* elhatározta, hogy a 2.6-os rendszermagba beépít egy *LISP* értelmezőt. A cél ezzel az, hogy lehetőséget teremtsen a rendszergazdáknak arra, hogy bizonyos biztonsági beállításokat egy magasszintű nyelven kezelhessenek a rendszermag módosítása és újrafordítása nélkül. *Szergej LISP* változata természetesen sokkal kisebb tudású, mint a *Common LISP*, így kevesebb memóriát foglal. Ráadásul egy önálló virtuális gépen fut, ami meggátolja, hogy az óvatlan rendszergazdák, vagy a *LISP* értelmező hibái kárt tehessenek a rendszerben. A legrosszabb, ami történhet, hogy a megadott biztonsági előírások egyszerűen nem működnek. *Szergej LISP* értelmezőjének gyakorlatilag nulla az esélye arra, hogy tényleg bekerüljön a rendszermagba. Ennek több oka is van. Először is többeknek erős a gyanúja, hogy a *LISP* nem lenne a rendszergazdák kedvence. Ők inkább valami héjszerűt szeretnének. Másodszor ugyanezt az értelmezőt a felhasználói térben is meg lehet valósítani, minek utána teljesen fölösleges azt beépíteni a magba. Ráadásul ez a megoldás a virtuális gépet is fölöslegessé teszi. Nem kizárt persze, hogy ez a projekt a rendszermagon kívül még sokáig fog élni, hiszen szerte a világon számos olyan lelkiismeretes programozó létezhet, akik szeretnek ilyesmivel játszani csak úgy, a móka kedvéért. (Vagy tényleg komolyan gondolják...)

Amúgy a 2.6-os rendszermag egész fejlesztési folyamata valahogy szokatlan, még linuxos mértékkel is. Kezdetben úgy tűnt, hogy *Linus Torvalds* teljesen átadta a koordinálást *Andrew Mortonnak*, de aztán mégis visszatért, és ma együtt dolgoznak. *Andrew* ugyanakkor tovább folytatja a 2.6.6-rc1-mm2 sorozat fejlesztését, amelynek nevében az *-mm* azokra az időkre emlékeztet, amikor a legtöbb problémát a memóriakezelés jelentette. Újabban megjelent egy -mc (merge candidate) sorozat is, amelynek célja a *Linus* által karbantartott forrásfával való egyesülés. A -mm sorozat újabb tagjai eközben folyamatosan keletkeznek, bár gyanítható, hogy végül ezeknek is az egyesítés lesz a sorsa. A felhasználók szempontjából minden a legnagyobb rendben van, hiszen a 2.6.5, 2.6.6 és 2.6.7 változatok a szokásos időközönként követték egymást. Ugyanakkor nyugodtan állíthatjuk, hogy a 2.6-os fa fejlesztése a szokásosnál zegzugosabbra sikerült.

Zack Brown (Linux Journal 125. szám)

© Kiskapu Kft. Minden jog fenntartva



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

© Kiskapu Kft. Minden jog fenntartva

A hónap szakmai tanácsai

Vezeték nélküli beállítások a Fedora-ban

A *Fedora Core 2* alatt van egy működő vezeték nélküli hálózati kártyám, és miután az `iwconfig` programmal elvégzem a beállításokat, képes is vagyok a hálózattal történő kapcsolattartásra. Azonban újraindítva a rendszert az összes beállítás elvész, minden információt újra be kell gépelnem. Van valami, amit elmulasztok, vagy esetleg máshol kell keresnem a hiba okát?
Weoh G, ➲ weoh3@cox.net

Az eszköz beállításához a rendszerbeállító eszközöket kell használnod: Task Bar (tálca) > System Settings (rendszerbeállítások) > Network (hálózat)
Christopher Wingert,
➲ cwingert@qualcomm.com

A vezeték nélküli eszközök parancssoros segédprogramjainak (`iwconfig`, `iwspy`, `iwpriv`) hatása nem marad meg a rendszer újraindítása után, ahogy te is tapasztaltad. A megoldás a beállítás megfelelő beállítófájlon keresztül történő megvalósítása.
A *Fedora* esetében nagy valószínűséggel állandósíthatjuk a szükséges beállításokat azzal, hogy hozzáadjuk a vezeték nélküli eszközre vonatkozó bejegyzéseket az ***ifcfg*** fájlunkhoz (***/etc/sysconfig/network-scripts/ifcfg-ethX***, ahol az ethX a vezeték nélküli kártyánkhoz rendelt eszköznév). Egyszerűen be kell írnunk a vezeték nélküli eszközre vonatkozó beállításokat ebbe a fájlba. Az én táskagépemen például, amely egy 802.11b felületen keresztül csatlakozik az útválasztó/tűzfal/dhcp kiszolgálóhoz a házamban, ez a fájl nagyon egyszerű:

```
DEVICE=eth1
BOOTPROTO=dhcp
ONBOOT=no
MODE=Ad-Hoc
CHANNEL=1
KEY=XXXXXXXXXX
```

Nekem mindössze a MODE, CHANNEL és KEY beállításokra volt szükségem az üzembe helyezéshez. A tiéd bizonyára különbözik ettől, de a parancssoros felületen keresztül elvégzett minden művelethez léteznie kell egy beállítási lehetőségnek a fájlban. A használható beállítási lehetőségek listáját az ***/etc/sysconfig/network-scripts/ ifup-wireless*** fájlban találjuk.
Timothy Hamlin, ➲ thamlin@nmt.edu

Az alábbi címen konkrét útbaigazítást találhatunk a vezeték nélküli hálózati eszközök ***Fedora Core Linux*** disztribúció alatti beállításaira vonatkozóan:
➲ www.siliconvalleyccie.com/linux-hn/wmp11-linux.htm.
Felipe Barousse Boué, ➲ fbarousse@piensa.com

A ***Linux Journal „Embedded Linux Journal"*** (beágyazott ***Linux Journal***) című 9. számában volt egy „Update on Single-Board Computers" című cikk, amelyben az egyik kép alatt ez volt olvasható: „An EBX Form Factor PowerPC-based SBC from ***Motorola*** " (egy EBX méretű PowerPC alapú SBC a ***Motorola***-tól). A kérdésem az, hogy melyik cég gyárt ilyen kártyákat: EBX méretű ***PowerPC*** alapú SBC.
Chirag Cheema, ➲ cscheema@tpcsed.com

Egy kis keresgéléssel két lehetőséget is találtam:

➲ www.embeddedplanet.com/products/rpxc.asp és

➲ www.acttechnico.com/mot-ebx-motorola.html.
Chad Robinson, ➲ chad@lucubration.com

Próbáld meg az ***Ampro*** nevű céget:
➲ www.ampro.com.
Felipe Barousse Boué, ➲ fbarousse@piensa.com

Rendszerinformáció lekérdezése

A gépek felépítésére vonatkozóan egy MACHINE_STATIC nevű rendszer által meghatározott szerkezetet használunk, amellyel meghatározhatjuk egy gép IP-címét, processzorának sebességét, az operációs rendszert és így tovább. Hasonló szerkezetre lenne szükségem Linux alatt is a gépek felépítésének meghatározására. Tud valaki ilyet?
Rajesh Kumar Patnaik, ➲ rajeshp_80@yahoo.co.in

Önmagában egyik sem egy szerkezet, de bizonyos `ioctl()` hívások alkalmasak sok ehhez hasonló adat meghatározására. Érdemes az új eljárásokat is megvizsgálni az információ kinyerésére, a procfs-t és a megjelenés előtt álló sysfs-t.
A */proc* könyvtárban lévő fájlok olvasása sok fontos rendszeradatot eredményezhet. Például a ***/proc/cpuinfo*** a processzor adatait tartalmazza, a ***/proc/net/route*** a hálózati útvonalakat (az IP-címek hexadecimális formában vannak feltüntetve), a ***/proc/version*** pedig a rendszermag verziószámáról tájékoztat.
Chad Robinson, ➲ chad@lucubration.com

Lemezjavító eszközök?

Tudja valaki, hogy ***Red Hat*** 8 alatt milyen eszközöket használhatunk lemez- és fájlhibák felderítésére és javítására (`scandisk, scan registry`)?
moo, ➲ moochoo_86@hotmail.com

Ezek az eszközök nem a használt rendszercsomag fajtájától, hanem a fájlrendszer típusától függnek. A többi rendszercsomaghoz hasonlóan a Red Hat

© Kiskapu Kft. Minden jog fenntartva

A másik gépem egy szuperszámítógép

Steve Jones 2002 novemberében kezdett el jegyzeteket készíteni a PBS-ről, MPI-ről és a Mosixról, majd 2003 júniusában már egy a TOP 500-as listán szereplő számítógéptelep vezetője volt. Jó példa ez arra, hogyan képes segíteni a Rocks-terjesztés egy telep vezetőjét a rendszer felállításában és üzemeltetésében.

Úgy két éve ***Mitch Davis*** (Stanfordi Egyetem műszaki tudományos vezérigazgatója) és ***Carnet Williams*** (a Stanfordi Egyetem műszaki tudományos igazgatója) egy igen komoly és nagy jelentőséggel bíró projekt kapcsán hívtak fel. A Stanfordi Jogi Egyetem hálózati műveletek osztályának vezetőjeként nagy örömömre szolgált együtt dolgozni Mitchel és Carnettel. (Mitch egyben a Jogi Kar dékánja is volt, Carnet pedig a főinformatikusi posztot töltött be ugyanitt.)

Ők meséltek nekem először ***Dr. Vijay Pande***-ről, a ***Folding@home*** projekt vezető kutatójáról, aki egy nagy géptelepet szeretett volna vásárolni. Megadták neki a nevemet, mivel olyan embert keresett, aki ezt a projektet képes sikerre vinni. Ösztönösen igent mondtam. Megbeszéltük a projekt részleteit és mielőtt leraktam volna a kagylót azért rákérdeztem: „Mekkora is lesz pontosan?" „300 kétprocesszoros csomópont" – válaszolták. „600 CPU ... Ez aztán tud tarolni" – gondoltam akkor.

Miközben Mitch, Carnet és Vijay a Dellel és az Intellel együtt a fürt megvásárlásáról tárgyaltak, én küldtem egy e-mailt Vijay Pande-nak, melyben leírtam, mivel tudnék segíteni neki a projekt hálózati és hardver részében. Egyben reményemet fejeztem ki, hogy az általa alkalmazni kívánt szoftvert a kivitelezés során jobban megismerhetem majd. Üzenetem utolsó sora a következő volt: „Valami nagynak szeretnék részese lenni ". Vijay azonnal válaszolt, és segítségemet örömmel fogadta. Megszerveztük első megbeszélésünket, mely során megvitattuk a projekt hatókörét.

Azon az első találkozón úgy tűnt, hogy a legtöbb dolog a levegőben lóg. Mindenki tudta, hogy jön a berendezés, de nem voltak valódi tervek. Vijay azt mondta, tisztában van vele, hogy a próbaüzemről és a használni kívánt fájlrendszerről dönteni kell. Emellett annak a lehetőségét is meg kellett fontolni, hogyan használhatjuk a már meglévő stanfordi szolgáltatásokat.

Vijay megemlítette a PBS, MPI és Mosix futtatásának lehetőségét is. Ezekről nagyon keveset tudtam, de feljegyzéseket készítettem és rákerestem a Google-lal a fentiekre, valamint a „beowulf" és „fürt" szavakra is. Belefutottam egy bemutatóba, ami a fürtépítésről, és egy Rocks nevű nyílt forráskódú szoftverről szólt. Utóbbit az NPACI szervezet készítette (➲ www.rocksclusters.org). Kitűnő bemutató volt, rengeteg kérdésemre azonnal választ adott. Ilyen volt többek között az az „egyszerű" probléma, hogy tulajdonképpen hogyan is kell összerakni egy ilyen fürtöt, hogyan kezeljük a csomópontokon futó szoftvereket, hogyan állítsuk be a mester-csomópontot és hogyan felügyelhetjük a többit. A bemutató gyakorlatilag megadta a vázát a mi tervezett fürtünk felépítésének. Kinyomtattam és magammal vittem a következő megbeszélésünkre. A készen kapott megoldás jó fogadtatásra talált.

1. kép Iceberg a Forsythe Adatközpontban

© Kiskapu Kft. Minden jog fenntartva

Folding@home az Icebergen

Az Iceberg ellenőrzi a Folding@home adatait. Ez egy elosztott számítási projekt, amelynek célja a fehérjék helyes és téves összekapcsolódásának, valamint az erre visszavezethető betegségek tanulmányozása. A megvalósításhoz önkéntesek szabad processzoridővel járulnak hozzá. Jelenleg körülbelül 80,000 processzor számolja az adatokat.
A Iceberget a Folding@home-mal kapcsolatos kutatások során kisebb feladatok szimulálására használom. A feladat itt egy olyan szimulációsorozat végrehajtását jelenti, amelyben egy fehérje egy előre meghatározott módon kapcsolódik össze egy másikkal. A kulcs egy olyan szkript megírása volt, amely leutánozza azt a folyamatot amelyben a Folding@home szétosztja a feladatot az ügyfeleknek. Egy futtatáshoz általában 10-20 CPU-t használok egyszerre.
Más, nagyobb projektekhez az Iceberget csak a kezdeti felosztáshoz használtam. Általában 10-50ns-os szimulációkat hajtunk végre 1ns-os darabokban. Az Iceberget az első 1ns-hoz használjuk, majd áttérünk a Folding@home-ra és ott folytatjuk a munkát. Új módszereinket gyorsan meg tudjuk ismételni az Iceberg által kínált kontrollált és stabil környezetben. Új projektek fejlesztésénél az Iceberget az eredmények ellenőrzésére használjuk, és amint biztosak vagyunk az új módszerben, ráeresztjük azt a 80,000 CPU-s elosztott számítógépre.

–Young Min Rhee a Folding@home projektből, folding.stanford.edu

Az Iceberg felállítása

Miközben ez a két találkozó zajlott, a Dell a Stanfordi Forsythe Adatközpontban a fürtöt szekrénybe szerelte és összekötötte a csomópontokat. Mindez hét napig tartott. Letöltöttem a Rocks 2.3 változatát, és a telepítettem a Rocks-zsargonban központnak nevezett gépre. Az egész valahogy elbűvölően egyszerű volt. A sikeren felbuzdulva harmadik megbeszélésünk után úgy döntöttünk, hogy a hardver és a hálózat felépítése mellett feladatköröm kibővült a szoftver kezelésével is. Biztos voltam benne, hogy a többi akadályt is sikerrel veszem majd, de utólag azt kell mondanom, hogy ezen a ponton még egyáltalán nem voltam tisztában a feladat valódi méreteivel. Valójában ugyanis minden idők legnagyobb Rocks fürtjét kezdtem el építeni.
Az első problémával akkor szembesültem, amikor megpróbáltam telepíteni egy számítási csomópontot. Erre való az `insert-ethers` nevű Rocks segédeszköz, amely a számítási csomópontok Ethernet MAC címeit megtalálja, IP címeket és gépneveket oszt ki nekik, majd ezt az információt – a PXE-t és a DHCP-t használva – egy megadott egyeztetési protokoll segítségével beilleszti egy adatbázisba. A csomópont beillesztését követően, az azon futó rendszer egy megfelelő Red Hat Kickstart állomány alapján épül fel és kerül beállításra, befejezve ezáltal a PXE rendszerbetöltési folyamatot.
Sajnos gondjaim adódtak a Dell PowerEdge 2650-ben található hálózati kártyával. Úgy tűnt a Rocks nem támogatja a Broadcom Ethernet vezérlőket. Elküldtem a problémámat a Rocks vitafórumára és a Dell-t is felhívtam támogatást kérve, és mivel arany fokozatú támogatási szerződésünk volt, nyitottam egy szervizjegyet.
A Rocks fejlesztői gyorsan készítettek számomra egy kísérleti változatot, amely tartalmazta a szükséges frissített eszközmeghajtókat. Ez megoldotta a problémát, és hamarosan viszontláttam a javaslataimat és észrevételeimet a Rocks 2.3.1 szerviz kiadásában.
Az utolsó gond amit a méretezésnél vettem észre, az volt, hogy nem voltam képes 511 aktív feladatnál többet futtatni. A felhasználóim sikítoztak a 100 kihasználatlan processzort látván. Ez a helyzet azért állhatott elő, mert az Icebergen futtatott feladatok nagy része rövid életű, egy-két processzoros folyamat volt. Miközben a Rocks fejlesztői csapatával dolgoztam együtt, megpróbáltunk előre meghatározott állandókat találni a Maui időzítő kódjában. Végül én találtam meg, és a Rocks csapatának útmutatásával, újrafordítottam és újraindítottam a Maui-t. A rendszer most már annyi aktív feladatot tud időzíteni, ahány szabad processzor van.

A TOP500-as futtatás

2002 végén, miután megoldottuk az utolsó hardver és szoftverproblémát is elhatároztuk, hogy az Iceberget feltesszük a TOP500 szuperszámítógép listára (➲ www.top500.org). A TOP500 egy félévente megrendezett verseny, amely a Linpack (egy lineáris algebrai feladatok megoldására használható csomag) tartós futtatásával mért teljesítmény alapján rangsorolja az 500 leggyorsabb gépet. A 2002 novemberi listán 97 hagyományos gépekből álló fürt szerepelt, így biztosak voltunk benne, hogy az Icebergnek van esélye a listára való felkerülésre.
A TOP500 listára való felkerülés azonban több munka volt, mint amire számítottunk. A Rocks egy előre lefordított Linpack állománnyal érkezik, amely képes ugyan jó teljesítményt elérni, de mi többet akartunk.
Kapcsolatba léptem a Dell Skálázható Rendszerek csoportjával. Tőlük a Linpack fürtre hangolásában kaptam segítséget. Dolgoztunk együtt például a Linpacknak a Goto BLAS könyvtárhoz való hozzáillesztésén (Basic Linear Algebr Subroutines; alapvető lineáris algebrai szubrutinok). Utóbbit ***Kazushige Goto*** írta (➲ www.cs.utexas.edu/users/flame/goto).
Ezen felül a Dell javaslatot tett egy kifinomultabb hálózati topológiára. A TOP500 futtatás előtt a 300 csomópont 16 db 100Mbit-es Ethernet kapcsolón osztozott (Dell PowerConnect 3024). Úgy találtuk, hogy a Linpacknak – mint megannyi más, erősen párhuzamosított alkalmazásnak – kifejezetten használ, ha egy jobb hálózati kapcsolatot építünk ki. (Magyarul kisebb késleltetési időt és/vagy nagyobb sávszélességet biztosítunk). A Dell kölcsönzött nekünk néhány Gigabites blokkolásmentes kapcsolót.
A fenti fejlesztések javították a teljesítményt, és 2003 júniusában bemutattuk eredményeinket a TOP500 listán. Az Iceberg a 319. helyen áll és érzésem szerint egy gyorsabb hálózati kapcsolattal akár előrébb is lehetne.

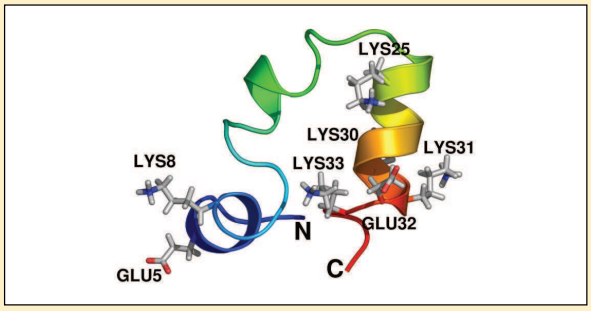
© Kiskapu Kft. Minden jog fenntartva

IN SILICO (számítógépes) biológia az Icebergen

Az Iceberg Dell szuperfürt létrehozása felért egy tudományos forradalommal. Tulajdonképpen évtizedek óta használunk szuperszámítógépes központokat, de mindig kényelmetlenül éreztük magunkat a meglehetősen szűkre szabott feldolgozási sorok, a kevés gépidő miatt, no meg azért, mert valahogy mindig nehézkesen lehetett csak a rendszert az igényeinkhez igazítani. Az elmúlt néhány évben felépítettük saját Linux fürtjeinket 50-100 CPU-val. A kész hardver azonban sajnos magasabb támogatási és adminisztrációs költségekhez vezet, arról nem is beszélve, hogy amíg cikket írunk, a fürt tétlenül áll. Az új Stanfordi Dell fürt ezzel szemben rendkívül költséghatékony megoldás kínál tudományos feladatokra. Megosztott erőforrásként csomópontok százai érhetőek el, ha mondjuk egyik éjszaka tesztelni szeretnénk egy új modellt, a költségünk viszont kizárólag az általunk használt átlagos csomópontszámmal arányos. Amellett, hogy a hardver teljes irányításunk alatt áll, az igénybe vehető erőforrások egy nagyságrenddel nagyobbak annál, mint amit valaha is használtunk a telepítés előtt vagy után. Ehhez képest az adminisztrációval hetente mindössze egy órát töltünk.

A számítógépes biológia fehérjekutatással kapcsolatos alkalmazásai általában rendkívül nagy számítási kapacitást igényelnek. Az egyik legáltalánosabb esetben az a feladat, hogy logikai összefüggést, feltűnő mintázatot találjunk a szekvenciacsaládok és a szerkezetek között. Hasonló, de kicsit más a biomolekulák belső mozgásainak szimulációja. A mintaegyeztetés nem nagy dolog, ha csak néhány tucat szekvencia áll rendelkezésünkre, a jelenlegi projektünkben azonban az cél, hogy a Shewanella baktérium fehérjéivel kapcsolatban tudjunk pontos előrejelzéseket adni. (Ez a baktériumfaj arról a képességéről híres, hogy radioaktív és mérgező hulladékot eszik). Ez pedig több százezer szekvenciát jelent, amelyeket páronként össze kell hasonlítanunk.

A helyi fürtünkön ez több heti alaposan megtervezett futtatást kívánt. Ugyanezt a jelenleg rendelkezésünkre álló eszközökkel egyetlen éjszaka alatt megtehetjük. Ennyi idő kell ahhoz, hogy kipróbáljunk egy új ötletet, másnap pedig már bemutathatjuk az eredményt.



1. ábra A Villin headpiece

A Villin headpiece önmagában egy nagyon kicsi, körülbelül 600 atomból álló protein. Ugyanakkor mindig víz veszi körül (vörös/fehér pálcikák), ami a kezelendő atomok számát körülbelül 10,000-re növeli. Minden egyes atom a legközelebbi 100-200 szomszédjával lép kölcsönhatásba. Ezeknek a hatásoknak az erősségét minden egyes lépésben ki kell számolni, majd ezt félmilliárdszor meg kell ismételni ahhoz, hogy a molekula mozgásának egy mikroszekundumnyi részletét pontosan leutánozzuk. Az 1. ábrához használt adatok az Iceberg 10 csomópontján két hetes futtatással készültek.

Az atomok mozgásának molekuladinamikai szimulációja szintén lenyűgöző feladat. Az ötlet nagyon egyszerű, ki kell számítani az atomok egymásra kifejtett erejét, majd Newton egyenlete alapján meg kell határozni az új pozíciókat egy megfelelően rövid idővel később (egy lépés általában 2 femtomásodperc, vagyis 2 × 10-15 másodperc). A szimuláció egy lépése ugyan gyors, egy biológiai reakció tanulmányozásához azonban lépések milliárdjai szükségesek . Ezért a kódot úgy optimalizáltuk, hogy kihasználhassuk a Pentium 4 Xeon processzorokon elérhető Intel Streaming SIMD Extensions utasításkészletet. Ezzel átlagosan kétszeres illetve négyszeres közötti gyorsulást érhetünk el. Ez tett elehetővé, hogy olyan méretű fehérjéket szimuláljunk több mint egy mikromásodpercnek megfelelő valós ideig, mint a Villin headpiece (1. ábra), s mindez csupán 2 hétbe telt a Iceberg 10 csomópontján. Valójában az optimalizált kód még egy önálló Dell/Intel Xeon processzoron is gyorsabb, mint a csúcskategóriás IBM Power4 vagy Alpha processzorokon, s mindezt tizedannyi költség mellett érjük el. Eddig ez volt messze a legjobb informatikai beruházásunk, így ha a jövőben bővítenünk kell, nem fogunk habozni.

– Michael Levitt és Erik Lindahl a Szerkezeti biológia tanszékről, Stanfordi Egyetem Orvosi Kar

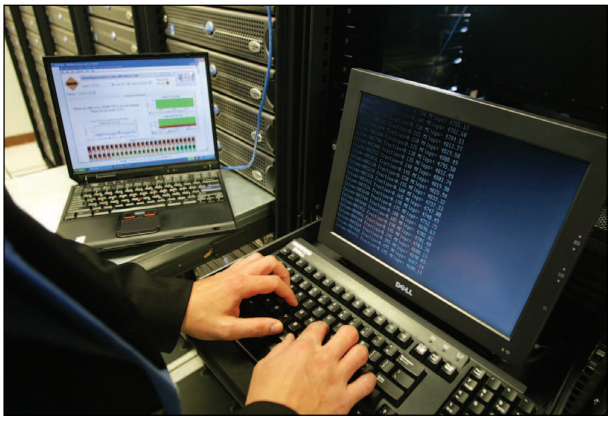
Az Iceberg új otthonába költözik

Az Iceberg állandó lakóhelyének a James H. Clark Központot szemeltük ki. Ez a Silicon Graphics és Netscape alapítójáról kapta a nevét, aki egyben a Bio-X Projekt alaptőkéjét szolgáltató mecénás is. 2003 augusztusában eljött a költözés ideje. A Forsythe Adatközpontból való elköltözés sok jó dolgot hozott számunkra, melyek közül az egyik legfontosabb a Rocks újratelepítése volt. Azért erőltettem ezt a dolgot, mert egy ekkora méretű fürt karbantartásának a stabil infrastruktúra a kulcsa. Ez teszi lehetővé a teljes tulajdonlási költség alacsonyan tartását. Amíg az Iceberg a Forsythe Adatközpontban volt arra a következtetésre jutottunk, hogy a hardveren és a szoftveren egyaránt van még mit javítani.

A költözés kapóra jött, hiszen az ezzel együtt járó állásidő lehetővé tette számunkra, hogy a módosításokat elvégezzük a fizikai modellen. Úgy döntöttünk, hogy a központi csomópontot továbbfejlesztjük, a felhasználói könyvtárakat pedig áthelyezzük egy másik, háttértárolóval rendelkező csomópontra.

A Rocks újra beváltotta a hozzá fűzött reményeket. Annyi volt csak a dolgunk, hogy az `insert-ethers` programban kiválasszuk a beillesztendő csomópont NAS készüléktípusát. Hogy teljes mértékben kiaknázhassuk az ebben található dupla Gigabit Ethernet hálózati kártyát, a kapcsolat-összefűzést használtuk. A központi csomóponton végzett néhány módosítás után, ami gyakorlatilag

© Kiskapu Kft. Minden jog fenntartva



2. kép Linpack futtatása a TOP500-ba való kerüléshez

3. kép Az Iceberg új otthonában
Balról jobbra: Vijay Pande, a Folding@home vezető kutatója; Steve Jones, az Iceberg tervezője; Erik Lindahl, posztdoktori ösztöndíjas; és Young Min Rhee, kutató

a felhasználók új készülékhez való hozzárendelését jelentette az előzőleg mentett adatok visszamásolása után újra működtünk.

A személyi szuperszámítógép filozófiája

Minden nagy teljesítményű számítógépfürt telepítésénél az elsődleges cél az, hogy azt a lehető leggyorsabban megbízhatóan működő állapotba hozzuk, és ez az állapot a hardver haláláig fenn is maradjon. A rendszert használó kutatók így biztosak lehetnek benne, hogy amikor egy problémával kapcsolatban nagyobb számítási teljesítményre van szükségük, az rendelkezésükre is áll. Egy ilyen méretű és kihasználtságú rendszert természetesen nem lehet csak a próba kedvéért módosítgatni.

Meglátásom szerint ezzel a koncepcióval tökéletesen összhangban van mindaz, amit az általunk választott standardizált fürt disztribúció biztosít. A rendszerfelügyelet ennek használatával kimerül a feldolgozási sorok illetve a fájlrendszerek telítettségének megfigyelésében. Szemmel kell tartani persze a különböző naplókat illetve magát a hardvert is, de ez tulajdonképpen természetes.

Kombinálva a Rocks és a Dell támogatási tervét a számítási pontokra ezüst, a központi csomópontra pedig arany fokozatú támogatást kaptunk, ami azt jelenti, hogy három évig nem lesz gondunk sem a teljesítményre, sem a meghibásodott alkatrészekre. A további finanszírozással kapcsolatban úgy gondoltuk, hogy a gépidő kiszámlázásával elő tudjuk teremteni azt az összeget, amellyel három éven belül lecserélhetjük a teljes rendszert. Erre aztán áttelepítjük a Rocks rendszert, és további három évig megint nem lesz problémánk.

Az Iceberg-II várhatóan ugyanannyi csomópontot tartalmaz majd, mint elődje, de a 2U helyett 1U magas gépek lesznek benne a helytakarékosság végett. Az elód különálló fürtként tovább működik majd, de egyre kevesebb csomóponttal, mert a meghibásodott elemeket már nem cseréjük benne.

Emellett tervezzük egy másik, 600 csomópontot számláló fürt beszerzését is, amit az elkövetkező 6-12 hónapban akarunk elindítani. Ez a fürt egy külső telephelyen lakik majd. Pillanatnyilag tárgyalunk egy kutatóintézettel, amely érdeklődött a lehetőség iránt. Ők megfelelő gépidőért cserébe speciális szolgáltatásokat (generált áram, szünetmentes táp, légkondicionálás) kínálnak.

Azt is tervezem, hogy az Iceberg mellett építek még egy fürtöt, kifejezetten kereskedelmi célokra. Szeretném, ha befejezése után nemcsak ez lenne a legnagyobb Rocks fürt, hanem bekerülne a TOP500 lista első 20 helyezettje közé.

Záró megjegyzések

Az Iceberg karbantartási költségeinek alacsonyan tartását alapvetően az teszi lehetővé, hogy körülötte éppen jó arányban vannak jelen az infrastruktúrával és a tényleges használattal foglalkozó emberek. Utóbbiak értelemszerűen azok a kutatók, akik számítási feladataikat futtatják a rendszeren. A műszaki csapat nemcsak karbantartja a rendszert, hanem lehetővé teszi annak az oktatásban való felhasználását, illetve azt is, hogy az érdeklődők megismerhessék az általa nyújtott szolgáltatásokat.

A feladatokat ésszerűen elosztottuk, így a tűzfal karbantartása, az ütemező beállítása, a felhasználói adatok vagy a hardver karbantartása egy-egy ember munkaidejének csak egy nagyon kis részét veszik el. A rendszer karbantartására fordított idő hetente átlagosan egy óra, ami talán a legfényesebb bizonyítéka a jó tervezésnek. Az sem elhanyagolható persze, hogy egy 203 csomópontot számláló fürt tulajdonlási költsége megegyezik egy 10 csomóponttal rendelkezőével.

Linux Journal 2003. november, 115. szám

Steve Jones (stevejones@stanford.edu)
Azért adta fel a Stanfordi Egyetem Jogi Karán betöltött állását, hogy egy internetszolgáltató biztonsági stratégájaként folytassa pályafutását. Jelenleg tanácsadóként a biztonsági rendszer kialakításáért felel. Jelenleg épp átköltözni készül Main államba, ahol egy oktatási intézmény stratégája lesz, illetve egy másik vállalatot is akar alapítani. Szabadidejében, egy 302 csomópontból álló, Icebergnek nevezett fürt rendszergazdája.

© Kiskapu Kft. Minden jog fenntartva

A Mars-járművek irányítása

A NASA Spirit és Opportunity nevű Mars-járműveit irányító csoport asztali operációs rendszerként a Linuxot használja.

Amerikai idő szerint 2004 január 3-án reggel 8 óra 30 perckor a *Földről* küldött mintegy féltonnányi fém izzása ragyogta be a *Mars* egét. Hat perccel később az ejtőernyő és légzsákok kinyílása után a fémcsomag elérte a vörös bolygó felszínét, 28 felpattanással megtett mintegy 300 méteres távolságot, majd lassan gurulva megállt. A légzsákok leeresztettek, s ezzel láthatóvá vált az általa óvott gúla alakú fémdoboz, amely fokozatosan szétnyílva utat adott a benne lévő szerkezetnek, egy hatkerekű robot geológusnak, amelyet alkotói *Spirit* névre kereszteltek.
Így kezdődött a *Mars* három hónapig tartó felfedezése. A *NASA Jet Propulsion* laboratóriumának (*JPL*) több száz tudósból és mérnökből álló kutatócsoportja számára a szomszédos bolygónk felfedezésének új fejezete során a *Spirit* szolgáltatja a látást és az összecsukható karjának végén lévő szerszámokból álló eszköztárat. Az *Opportunity*, a *Spirit* ikertestvére három héttel később kezdi meg működését a *Mars* ellenkező oldalán. És vajon ezek a tudósok mit használnak a járművek irányítására? Nem mást, mint a *Linuxot*.
A *MER- küldetés* (*Mars Exploration Rover*, mars-felfedező jármű) fordulópontot jelent a *Linux* űrprogramokban történő felhasználásában. A *Linuxot* korábban is használták már űrkutatási programok során – például az *STS-83* űrrepülőgép fedélzetén már 1997-ben is *Debiannal* felszerelt hordozható gépet láthattunk –, de a *Mars-felfedező jármű* projekt az első *JPL*-küldetés, amely létfontosságú műveletek elvégzésére használja a *Linuxos* rendszereket. A *MER* során a *Linuxot* magasszintű tudományos tervezésre és alacsonyszintű parancselőállítási műveletekre, megjelenítésre és szimulációra egyaránt alkalmazzuk.

Hogyan jutottunk el idáig

Bármilyen furcsának tűnik is, eleinte nem állt szándékunkban a *Linuxot* használni elsődleges fejlesztési felületként. A programunkat eredetileg csak a 2001-es Mars-küldetésben akartuk felhasználni, amely lényegében a JPL 1997-es *Mars Pathfinder* küldetésének megismétlése lett volna. A terv az volt, hogy a jármű irányítását végző parancsokat egyetlen *Silicon Graphics* munkaállomásra bízzuk, ahogy a *Pathfinder* esetén is tettük korábban. Akkoriban a *Linuxos* kísérleteink gyenge próbálkozások voltak az *SGI* lehetőségeihez képest.

A *JPL* egy 98-as *Mars*-expedíciójának kudarca után azonban újragondolta a *Marssal* kapcsolatos stratégiáját. A 2001-es terveket egy későbbi indítás érdekében elvetettük, s ez a későbbi program lett végső soron a *MER*. Ebből kifolyólag a tervezettnél két évvel több fejlesztési idő állt rendelkezésünkre, jóllehet, nem is arra az űrhajóra vonatkozóan, ami az eredeti tervekben szerepelt. A *MER* járművei nagyobbak, intelligensebbek és összetettebbek, mint a *Pathfinder Sojourner* űrjárója, és mindegyik robotkarral is fel van szerelve.
Azokban az években a *Linux* és a futtatására alkalmazott x86 típusú gépek óriási ütemben fejlődtek mind a processzorsebesség, mind pedig a grafikus teljesítmény területén, s ez nem kis részben az *NVIDIA* grafikus lapkáinak és ezek erőteljes *Linuxos* támogatásának volt köszönhető. Ennek eredményeképpen kezdtünk egyre inkább a gyorsabb, jobb és olcsóbb Linux felé fordulni. A *MER* űrjárműveket így több csoport irányítja, párhuzamosan dolgozva a küldetéshez beszerzett több tucatnyi *Linuxos* gépen.

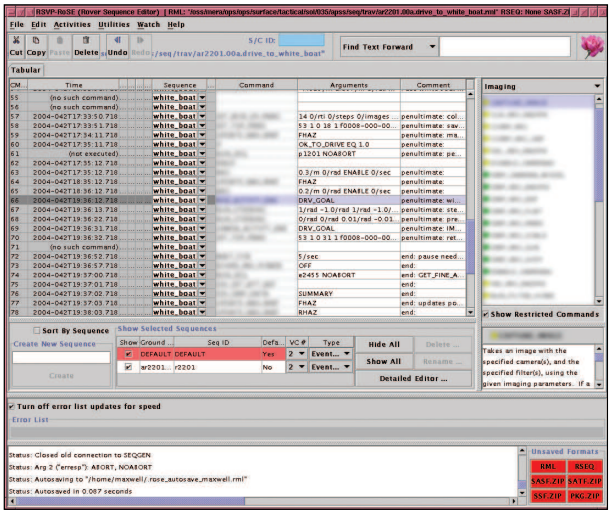
Az RSVP-rendszer

A használt *RVSP* (*Rover Sequencing and Visualization Program*, űrjármű vezérlő és megjelenítő program) nevű programcsomagunkat *Linux* alatt fejlesztettük, teszteltük és helyeztük üzembe. Az *RVSP* kifinomult eszközöket biztosít a *MER* tudósai és mérnökei számára a *Mars-járművek* irányításához. A nagymértékű fényidő-késleltetés miatt, amely a *Spirit* leszállásának napján közel 20 perces válaszidőt jelentett, lehetetlen a járműveket interaktív módon irányítani. Az *RVSP* ehelyett egy teljesen valósághű környezetet biztosít, amely teljes részlethűséggel jeleníti meg a jármű környezetét és szimulálja annak viselkedését. A marsi éjszaka folyamán az *RVSP* segítségével parancsszinten megtervezzük a napi teendőket, majd a kapott utasításokat műholdas kapcsolaton keresztül küldtük el a járműnek. A kapott parancsokat az űrjármű a következő marsi nap folyamán dolgozta fel. Az *RVSP* két fő alkotórésze a *RoSE* (*Rover Sequence Editor*, az űrjármű műveletszerkesztője), amely szöveges alapú felületet nyújt az űrjármű által végrehajtható parancsokhoz, valamint a *HyperDrive*, amely fejlett háromdimenziós grafikát szolgáltat a vezetési, karmozgatási és képalkotási parancsokhoz. A független programok mindegyike képes önálló működésre is, de igazából együttműködve hatékonyak. Együttes üzemmódban a programok az

© Kiskapu Kft. Minden jog fenntartva

Oak Ridge National Laboratories által kifejlesztett *PVM* (paralell virtual machine, párhuzamos virtuális gépek) program vezérlése alatt futnak. A *PVM* egy olyan adatbuszt biztosít, amely lehetőséget ad az *RVSP* komponenseinek a tevékenységük üzenetváltásokon keresztül történő összehangolására.

Az üzeneteink legtöbbjének tartalma az *RML*-re épül, amely az *XML* egy kifejezetten az űrjárművek parancssorozatainak leírására kidolgozott változata. Ennek az üzenetváltó megközelítésnek számos előnye van, amelyek közül nem elhanyagolható az sem, hogy az egyes különálló eszközöket különböző nyelveken valósíthatjuk meg. Amennyiben az eszköz képes *PVM*-üzenetek küldésére, máris a csomag tagjává válhat. A *RoSE*, az üzenettovábbító egység, és az üzenetnaplózó Java nyelven íródott; a HyperDrive, a képnézőnk és a parancsfolyam-böngészőnk forrásnyelve pedig a *C* és a *C++*. A következő küldetésben esetleg olyan parancsnyelveket is ki fogunk próbálni, mint a *Perl* és a *Python*.



1. kép A RoSE parancsszerkesztő (az ITAR-korlátozások miatt homályosítva)

A RoSE

A *RoSE* egy *Java* nyelven írt program, amely a *Sun Java* virtuális gépének 1.3.1 változatán fut, de nem a *Sun* fordítóprogramjával, hanem az *IBM* sokkal gyorsabb *Java* fordítójával, a *Jikes*-szal lett lefordítva. A *RoSE* kihasználja a *Java* jól ismert hordozhatóságát: jó hasznát vettük annak, hogy egy gyors *Linuxos* gépen fordíthattuk és tesztelhettük a programot, és semmi vagy nagyon kevés gondot okozott ennek az *SGI*-re történő telepítése és tesztelése. Még azzal is kísérleteztünk, hogy *Mac OS X*-re átültessük a programot, de ezzel felhagytunk, mivel nem volt arra időnk, hogy egy harmadik platformot is támogassunk. A *Mac OS X*-en sem jöttek elő kézzelfogható problémák, de ha fel is merültek volna, akkor sem jut időnk foglalkozni velük. Igény sem igen mutatkozott a *Mac OS X* támogatására, így teljesen elejtettük ezt a fejlesztési szálat.

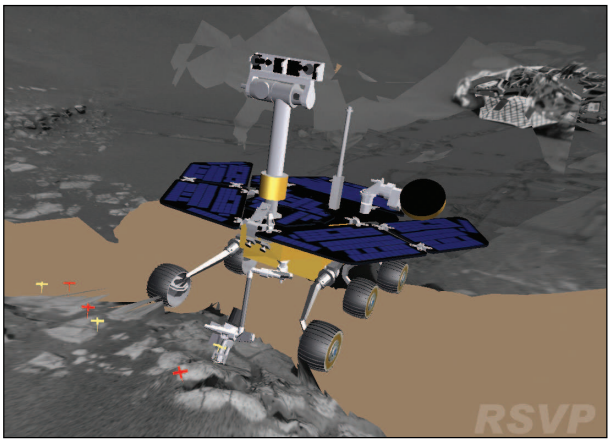
A *RoSE* igen erőteljesen adatvezérelt alkalmazás, amely az űrjárművel kapcsolatos minden tudnivalót *XML* állományokból olvas ki az indítási idő alatt. A Rose fejlesztéséhez használt integrált fejlesztői környezet a jól bevált *GNU Emacs* volt. A *RoSE* grafikus felületének nagy része a szabványos *Java GUI*-eszközkészlettel, a *Swinggel*, kézzel összerakott *Java*-kód. A grafikus felületet a *Swing* segítségével összerakni gyakran unalmas dolog. Ha a *Glade*-nek elérhető lett volna a *Javát* támogató változata, valószínűleg azt használtuk volna. A felhasználói felület fontos részei mégis dinamikusan kerülnek előállításra. Az indítási időben a *RoSE* által beolvasott egyik *XML* beállítófájl a jármű parancskönyvtárának a leírása, ami valami olyasmit jelent, mintha a jármű programozói felülete (API) lenne. A *RoSE* ezt a parancskönyvtárat használja arra, hogy dinamikus párbeszédablakokat állítson elő az egyes parancsok szerkesztésére, s ezzel a megközelítéssel rengeteg munkát takaríthattunk meg, mivel a programkönyvtár a fejlesztés évei során drámai mértékben megváltozott.

A *RoSE* megbízhatóságához jelentős mértékben járult hozzá a *Linux* könnyen automatizálható környezete. Már a kezdeti időszaktól kezdve terjedelmes öntesztelőkódokat írtunk a *RoSE* számára, majd egy egyszerű cron-munkafolyamatot hoztunk létre, amely minden éjjel lefuttatta ezeket a teszteket. Ha valamelyik teszt hibát jelzett, a következő reggel a fejlesztő erről elektronikus levélben kapott értesítést. Ezzel lehetővé vált a programhibák korai észlelése, amikor a hibát okozó változtatások még frissen éltek az emlékezetünkben. A *RoSE*-nak ez az intenzív öntesztje egyfajta kísérlet volt, s az ebbe fektetett energia később busásan megtérült.

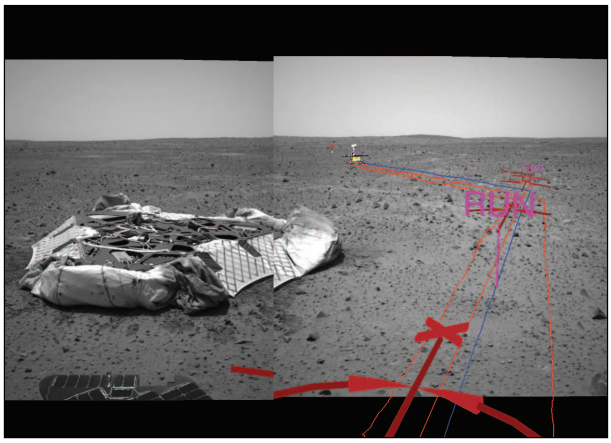
Mellékesen jegyzem meg, hogy a *Java*-t gyakran mondják lassúnak. A tapasztalataink alapján azonban azt kell mondanom, hogy ez a híresztelés teljességgel alaptalan, habár vannak a *RoSE*-nak olyan területei, ahol gyorsabb is lehetne, de ez bármelyik programról elmondható. Ha több időnk lett volna – ez a programfejlesztés állandóan emlegetett problémája – a program összes szűk keresztmetszetű részét sikerült volna kijavítani. A céljainknak a *Java* megfelelő sebességűnek bizonyult, a *Linux* jó *Java*-támogatása pedig nagyon nagy előnyt jelentett számunkra.

A HyperDrive

A *HyperDrive* az *RVSP* programcsomag interaktív megjelenítést végző összetevője, az a rész, amit majd a *CNN*-ben fognak mutogatni. A *HyperDrive* elsődleges célja, hogy lehetővé tegye a jármű irányításának és karmozgatásának biztonságos és hatékony megtervezését oly módon, hogy az irányítónak jól használható információkat ad a jármű pillanatnyi helyzetének és környezetének megértéséhez. Mindezt úgy éri el, hogy a rendelkezésre álló adatforrások egyesítésével egy háromdimenziós képet állít össze, és biztosítja ennek többféle megjelenítését. A *HyperDrive* számára az adatok egyik legfontosabb forrását a sztereó képek alapján előállított terepmodellek jelentik. A két járműre szerelt kamerák nagy része sztereó kamera, ami azt jelenti, hogy az emberi szemhez hasonlóan képesek a mélység érzékelésére. A sztereó összefüggésnek nevezett technika felhasználásával a képek sík, kétdimenziós világát háromdimenzióssá leképezett számítógépes grafikává alakíthatjuk át. A terepmodellek és a jármű által szolgáltatott CAD-adatok együttes felhasználásával előáll annak a rendszernek a magja, amely



2. kép Egy szikla árfúrása a kőkoptató eszközzel



3. kép A Spirit elhagyja a menedékét a kibővített élethűségű nézetben

© Kiskapu Kft. Minden jog fenntartva

képes a jármű nézetének leképezésére és lehetővé teszi annak interaktív mozgatását a marsi terepen, vagy a kar megfelelő mozgatását a bolygó felszínén (*2. kép*).
A leképezett képeket egyesítve a jármű kameráival nyert nyers képekkel egy még valósághűbb ábrázolásra nyílik mód, melynek során a jármű által érzékelt képet lefedve a mesterségesen előállított képpel, tanulmányozhatjuk azt, LCD shutter-szemüveggel akár három dimenzióban is (*3. kép*).
A jármű bármely kamerájának képe alapján szimulált nézetek állíthatók elő, segítve a kép alapján történő célkiválasztást. Ehhez a mesterségesen előállított világhoz olyan ikonokat és megjegyzéseket fűzünk, amelyek szemléltetik a végrehajtott parancssorozatot és leírják ennek a világnak a tulajdonságait. Ezeknek az objektumoknak az egyik legfontosabb csoportja egy ikonsorozat, amely a jármű által futtatandó parancsokat mutatja. Míg a ***RoSE*** lehetővé teszi a teljes parancssor szerkesztését, a ***HyperDrive*** csak azokat a parancsokat mutatja, amelyeknek érzékelhető vizuális hatásuk van. A ***HyperDrive***-ban a mozgatással, a robotkarral és a képalkotással kapcsolatos parancsok a terep felett megjelenő ikonsorokat képeznek, mindegyik azon a helyen, ahol végrehajtható, így lehetővé válik a jármű terepviszonyoktól függő mozgásának vizuális programozása és a robotkar nagy pontosságú célravezetése.
A ***Linux*** alatti háromdimenziós grafikai programozás egy addig egyedülálló kalandnak indult és a megbízható illesztőprogramokkal, programkönyvtárakkal egy alkalmas felületté érett. A ***HyperDrive*** a ***Silicon Graphics OpenGL Performer*** leképező programozói felületére épül.
A ***HyperDrive*** felhasználói felületének létrehozásához a GTK+ és libglade eszközöket használtuk. A ***Glade*** és libglade gyors prototípuskészítési lehetősége egyszerűvé és hatékonnyá tette a grafikus felület programozását. Mindenkinek bátran ajánljuk ezt az eszközkészletet, akinek megbízható felületkészítő eszközre van szüksége. Ahogy a fejlesztés célpontja az ***IRIX*** helyett minél inkább a ***Linux*** lett, egyre több nyílt forrású eszközt és programkönyvtárat használhattunk, majd a kompatibilitás megőrzése érdekében gondoskodtunk a létrehozott program visszaültetéséről ***IRIX*** alá.
A ***HyperDrive*** képes továbbá a járművek megjósolt viselkedésének animálására, az elképzelt mozgás valós időben történő megjelenítésére, lehetővé téve a felhasználó számára, hogy különféle nézőpontokból vizsgálja a jármű viselkedését.

Ugyanezt a képességét használhatjuk arra, hogy a marsjármű napi jelentése alapján visszajátsszuk a szerkezet valóságos működését.

Összegzés

Mindent összevetve, a ***Linux*** érett és a várakozásainkat messzemenően felülmúlva teljesítő programfejlesztő felület. A csapatunk tagjai közt a kezdetekkor előfordultak tapasztalt ***Linux***-támogatók és a témában teljesen járatlanok is, de mindnyájan elmondhatjuk, hogy hasznunkra voltak a ***Linuxszal*** és a felhasználói számára elérhető rengeteg nyílt forrású programmal kapcsolatos új tapasztalatok. Különösen az ***OpenGL***-alapú grafika az a terület, amely a drága grafikus szervereink egy alacsony kategóriájú alternatívájaként indulva végül a programjaink szempontjából felülmúlta a legjobb gépeket is.
Ezt a kutatást a ***Jet Propulsion*** laboratóriumban a ***Kaliforniai Műszaki Intézetben*** hajtottuk végre a ***NASA***-val kötött szerződés alapján. Semmilyen ebben a cikkben kereskedelmi termékre, folyamatra vagy szolgáltatásra a márkanévvel, márkajellel, a gyártó nevével vagy egyéb módon történő hivatkozás nem képezi vagy vonja maga után az ***Egyesült Államok*** kormányának, a ***Jet Propulsion*** laboratóriumnak vagy a ***Kaliforniai Műszaki Intézetnek*** a jóváhagyását.

Linux Journal 2004. szeptember, 125. szám

Frank Hartman jelenleg számítógépes grafikával és a Mars-űrjárművek vezérlésével foglalkozó szoftvermérnök a Jet Propulsion Laboratóriumban a kaliforniai Pasadenában. A Philadelphiai Művészeti Egyetemen szobrászatból szerzett képzőművészeti diplomát, és egy repülő- és űrhajózási mérnöki diplomát a Stanford Egyetemen. Frank a kaliforniai Altadenaban él feleségével, Leah-val.

Scott Maxwell programfejlesztő és a Mars-űrjármű irányító a Jet Propulsion Laboratóriumban a kaliforniai Pasadenában. Szeret aikidózni, klasszikus irodalmat olvasni és az életét röviden összefoglalni. Ő a Linux Core Kernel Commentary írója.

© Kiskapu Kft. Minden jog fenntartva

A GPS Toolkit

Ismerkedjünk meg a GPS működésével, és egy ingyenes könyvtár segítségével szerezzünk pontosabb adatokat földrajzi helyzetünkről!

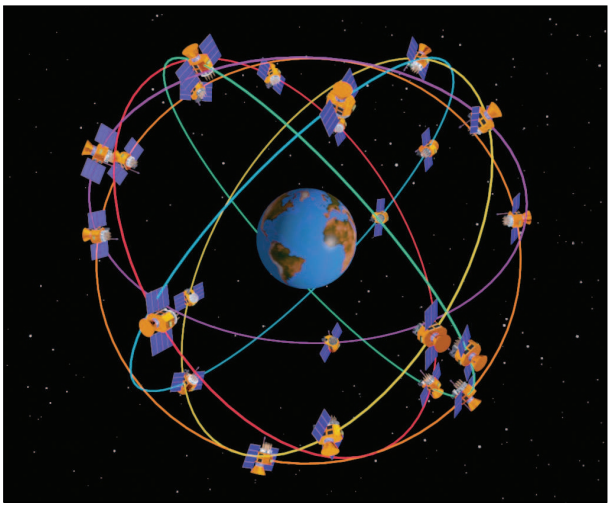
Robbanásszerű – talán ez a legjobb kifejezés a globális helymeghatározó rendszerrel (*Global Positioning System, GPS*) kapcsolatos piacon az elmúlt években látott növekedésre. A növekedés okai szerteágazók, de a legfontosabb talán a gazdasági, nevezetesen az, hogy a *GPS* elérése teljesen ingyenes, és az ehhez szükséges eszközök ára is meredeken zuhan. Ennek köszönhető, hogy a *GPS*-felhasználók helymeghatározásra képes készülékek széles választékából csemegézhetnek. A *GPS*-t régóta használják más területeken is, az űrbéli időjárás vagy a földrészek mozgásának vizsgálata és a pontos időzítési adatok szolgáltatása csak három példa ezek közül.
Ha a *GPS*-t komolyabb célokra, vagy akár csak pontosabb helymeghatározásra szeretnénk használni, akkor a *GPS*-vevő által begyűjtött nyers megfigyelési adatokat fel kell dolgoznunk. Korábban az ilyen feldolgozásokat jellemzően zárt programok végezték. *GPS Toolkit*, röviden *GPSTk* névvel azonban ma már létezik egy tervezet, amely a nyílt forrás és a kutatói közösségek számára *LGPL* hatálya alatt érhető el. A *GPSTk* az austini *Texasi Egyetem Alkalmazott Kutatások Laboratóriuma* által vezetett, még 1978-ban, az első műhold fellövése előtt indított *GPS*-vonatkozású kutatás melléktermeke. A munkában programmérnökök és tudósok egyaránt részt vesznek. A labor munkatársai nemrég úgy döntöttek, hogy alapszintű *GPS*-feldolgozó programjuk nagy részét nyílt forrással elérhetővé teszik – így született a *GPSTk*.

A globális helymeghatározó rendszer

A *GPS* valójában az *USA* kormányzatának polgári jelet is szolgáltató műholdas navigációs rendszere. Írásunk születésekor összesen 29 darab – 12 órás pályán keringő – műhold szórja jelét folyamatosan. A *Föld* bármely pontjáról minden pillanatban 8-12 műhold látható.

A GPS-ről röviden

Mindegyik műhold szórt spektrumú jeleket szór az 1575,42 és az 1227,6 MHz-es frekvencián, ezeket rendre L1-nek és L2-nek nevezzük. A polgári jel jelenleg csak az L1-en érhető el. A jel két összetevőből áll, az időkódból és a navigációs üzenetből. A kapott időkód és a belső időkód különbségét véve a vevő meg tudja határozni, hogy a jel mekkora távolságot tett meg. A kódok közötti különbséget a – nyilván nem tökéletes – vevőoldali óra hibái is befolyásolják,

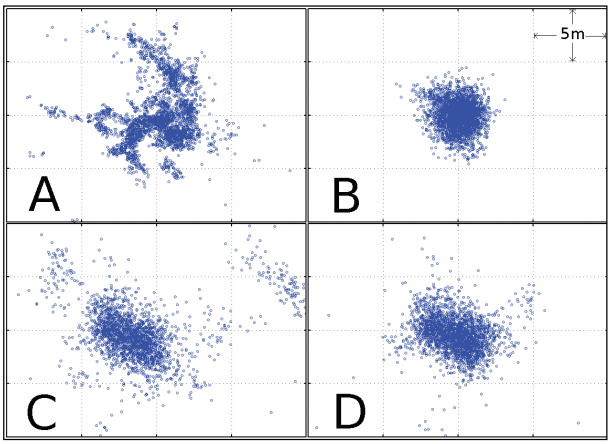


1. kép GPS műholdak együttállásának képe az Aerospace Corporation weboldaláról (➲ www.aero.org/news/current/gps-orbit.html)

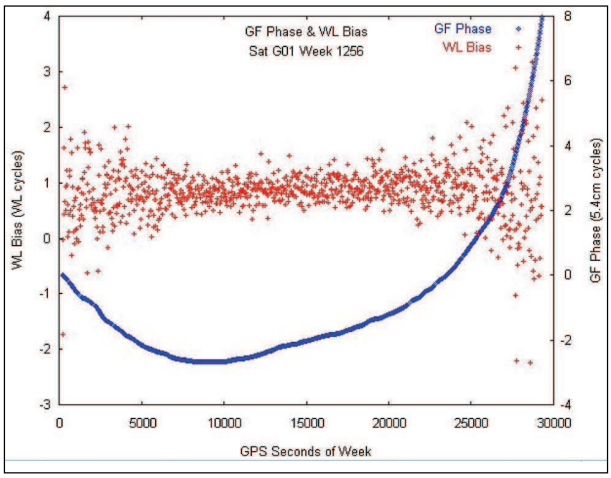
ezért általtartománynak hívjuk. A navigációs üzenet a műhold efemeriszét, vagyis pályájának numerikus modelljét tartalmazza.
A *GPS*-vevők az általtartomány mellett a vivőfázist, röviden a fázist is mérik, rögzítik. A fázis az általtartományhoz hasonlóan adott tartományba esik, ám benne ismeretlen konstans érték is szerepel, amit fázisbizonytalanságnak nevezünk. Itt már sokkal egyenletesebb értékről van szó, amely az általtartománnyal összevetve századannyi mérési zajtól terhes, így pontos helymeghatározásra is alkalmas. Mérésének módjából fakadóan a fázis véletlenszerű, hirtelen ugrásokat mutat. Ezek a diszkrét változások mindig a *GPS*-jel hullámhosszának többszörösei, és cikluscsúszásoknak nevezzük őket.

A helyzetmegoldás

Egy normál helyzetmegoldáshoz minden látható műholdról egy általtartomány mérésre és egy efemeriszre van szükség. Legalább négy mérésre van szükség, ugyanis négy ismeretlenünk van: három helykoordináta és a vevőoldali óra eltérése. A megoldás alapszintű algoritmusának ismertetése a hivatalos *GPS Interface Control Documentben*,



2. ábra Bal oldalon helyzetadatok egy GPS-vevőtől. Jobb oldalon a GPSTk algoritmusai által számított helyzetek.



3. ábra Normál szélessávú (vörös) és geometriamentes (kék) fázis egy műholdról

© Kiskapu Kft. Minden jog fenntartva

az ICD-GPS-200-ban található. A helyzetmegoldás pontosságát két tényező rontja, a megfigyelési és az efemeriszben észlelt hibák.

A mérési hibák csökkentése

A *GPS*-jel a Föld légkörének minden rétegén áthalad. A rétegek mindegyike más hatást gyakorol a jelre. Az ionoszféra, a légkör nagy magasságú, elektromosan töltött rétege például késlelteti a jelet, ami miatt nő a távmérési hiba. A késleltetés frekvenciafüggő, vagyis közvetlenül kiszámítható, ha mindkét *GPS*-frekvencián kapunk adatot. A troposzféra, a légkör legalsó rétege szintén késlelteti a jelet, ám ezt a késleltetést is lehet modellezni és semlegesíteni. Az egyéb hibák jelentős része magával a *GPS*-jellel kapcsolatos, ilyenek például a többszörös visszaverődések és a relativisztikus hatások.
A pontosabb alkalmazások a hibák hatásait az úgynevezett különbségi, differenciális *GPS* (*DGPS*) megoldással csökkentik. Ennél a felhasználó és egy közeli viszonyítási vevő által kapott jelek eltéréseit figyelve a mindkét vevő által érzékelt hibák, vagyis a hibák túlnyomó része kiküszöbölhető. A *DGPS* a viszonyítási vevőhöz képesti helyzetet ad meg, ehhez hozzáadva a viszonyítási helyzetet megkapjuk a felhasználó abszolút helyzetét.
A *DGPS* használatának alternatívája a hibák modellezése és közvetlen semlegesítése. A *GPS*-jeleket érintő hatások új és hibaforrásokra érzéketlen modelljének megalkotása komoly kutatások tárgya az ARL:UT-n és más laborokban egyaránt. Ilyen modellek kidolgozására kiindulásként a helymeghatározó algoritmus használható. Az alapszemlélet az, hogy a helymeghatározó algoritmust kifordítva meg lehet vizsgálni magukat a helyesbítő lépéseket. Ha például hálózatot állítunk össze a vevőkből, és összesítjük ezek megfigyeléseit, akkor világszintű térképet készíthetünk az ionoszféráról.

Továbbfejlesztett efemeriszek

A *GPS* alapú helyzetmegoldások pontossága leginkább jobb műholdas efemeriszekkel javítható. Az amerikai *National Geospatial-Intelligence Agency (NGA*) feladata a pontos efemeriszek előállítása és nyilvánossá tétele. Ezek alapján pontosabban meg lehet ismerni a műholdak pályáját. A szórt navigációs üzenetekben szereplő pályaadatok méteres nagyságrendű hibát tartalmaznak, a pontos efemeriszek ellenben 10 cm nagyságrenddel pontosak. Az *International GPS Service* (*IGS*) egy világméretű polgári együttműködés, amely szintén szolgáltat pontos efemeriszeket. A pontos efemeriszek előállításához szükséges adatokat megfigyelő állomások világméretű hálózatai szolgáltatják.

GPS adatforrások

Sok megfigyelőállomás adatai szabadon elérhetők az interneten keresztül, de az állomások jelentős része az *IGS*-nek is átadja eredményeit. Emellett sok hálózat szintén közzéteszi eredményeit az interneten, ilyen például az *Australian Regional GPS Network* (*ARGN*), illetve a *NASA*-hot kötődő, világszintű *Crust Dynamics Data Information System* (*CDIS*) rendszer.

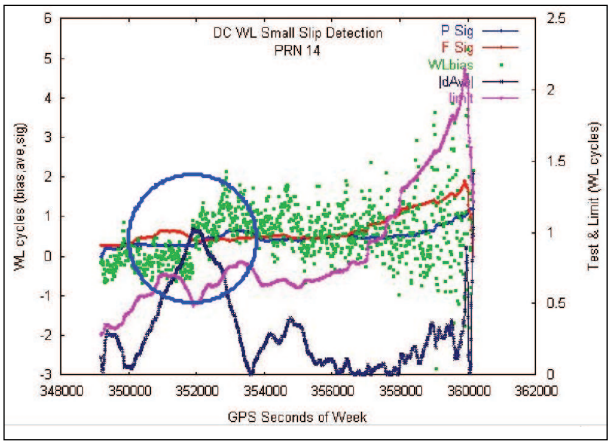
GPS fájlformátumok

A *GPS* megfigyelések eredményeit kutatók által, kutatók számára fejlesztett, szabványos formában szokták rögzíteni. A formátum kapcsán fontos gondolat, hogy az adatoknak függetleneknek kell lenniük a begyűjtésüket végző megfigyelőeszköz típusától. Ezt tükrözi a formátum neve is: receiver independent exchange (kb. vevőtől független adatcsere), röviden *RINEX*. Egy másik *GPS* vonatkozású formátum az *SP-3*, amely pontos efemeriszeket tárol. A *GPSTk* a *RINEX* és az *SP-3* formátumot egyaránt támogatja.

A GPS-vevők és a nyílt forrás viszonya

A *GPS*-vevők az elmúlt években egyre olcsóbbak és egyre nagyobb tudásúak lettek, különösen, ami a hordozható készülékeket illeti. A vevők szolgáltatásainak egy része általánosnak mondható. Például mindegyikük néhány másodpercenként bocsátja ki kimenetén az éppen érvényes helyzetmegoldást. A vevők mindegyike tárol egy helyzetlistát is, ennek elemeit útvonalpontoknak nevezzük. Sok készülék képes feltölthető térképek megjelenítésére, továbbá jelentős részük PC-vel vagy kézigéppel is tud kapcsolatot tartani abból a célból, hogy a gépen adattárolást végezzen vagy helyzetadatokat adjon át a rajta futó térképprogramnak.
A számítógépekkel és egyéb eszközökkel folytatott adatcsere általában a *National Marine Electronics Association*

© Kiskapu Kft. Minden jog fenntartva



4. ábra Csúszás észlelhető (kék kör) a szélessávú adatsorban (zöld), ahol a próbamennyiség (sötétkék) nagyobb a határértéknél (bíbor)

NMEA-0183 jelzésű szabványa szerint történik. Az ***NMEA-0183*** egy ASCII alapú, helyzetmegoldások, útvonalpontok és vevőoldali hibakeresési adatok továbbítására alkalmas formátumot ad meg. Egy sornyi NMEA formátumú adat, más megnevezéssel mondat, körülbelül így épül fel: $GPGLL,5133.81,N,00042.25,W*75

Az adatsor egy szélesség-hosszúság pontot határoz meg: 51° 33,81 perc észak, 0° 42,25 perc nyugat. A záró rész az ellenőrző összeg.
Nyilvános szabványként az ***NMEA-0183*** formátum biztosítja a választás szabadságát a GPS-használóknak. A nyílt forrású alkalmazások általában ***NMEA-0183*** formátumban fogadják a helyzetadatokat a vevőegységektől.
Zárt szabványokkal is gyakran találkozhatunk. A ***SiRF*** például zárt protokoll, használatának jogát a vevőegységek gyártói vásárolhatják meg, de nem egy gyártó saját bináris protokollt fejlesztett ki. Azóta ezeknek a protokolloknak egy részét megnyitották, néhányat pedig visszafejtettek.
A ***GPSBabel*** egy nyílt forrású tervezet, célja a fogyasztói szintű vevőkkel végzett adatcsere biztosítása. A ***Sharc Project*** hasonló tervezet, ám célja a földmérési szintű vevőkkel való adatcsere lehetővé tétele.
A fogyasztói szintű vevőkhöz számos figyelemre méltó nyílt forrású alkalmazás létezik. Van köztük autós navigációra használható, például a ***GPS Drive Project*** mindehhez grafikus térképet rajzol nekünk. A ***GPS Drive a Festival*** nevű alkalmazással is összecsatolható, így a vezetési utasításokat beszéd formájában kapjuk meg. A ➲ WiGLE.net és a hozzá hasonló oldalak a nyilvános vezeték nélküli hozzáférési pontok koordinátáit gyűjtik, ***GPS*** készülékünkkel könnyedén megtalálhatjuk őket.
A ***DGPS*** megvalósítása hagyományosan kettő vagy több vevővel történik, amelyek helyzetadataikat rádióhullámokon továbbítják. Nyílt forrású alkalmazásokkal a ***DGPS***-t immár IP felett is meg tudjuk valósítani, ugyanis a gpsd nevű nyílt forrású tervezet képes ***NMEA-0183*** mondatokat TCP/IP felett továbbítani. A ***gps3d Project***, amely helyzetünket és a ***GPS*** beállításait három dimenzióban jeleníti meg, szintén képes ***gps3d*** kiszolgáló segítségével dolgozni. Mindezen alkalmazások szabványos helymeghatározásra alapulnak.

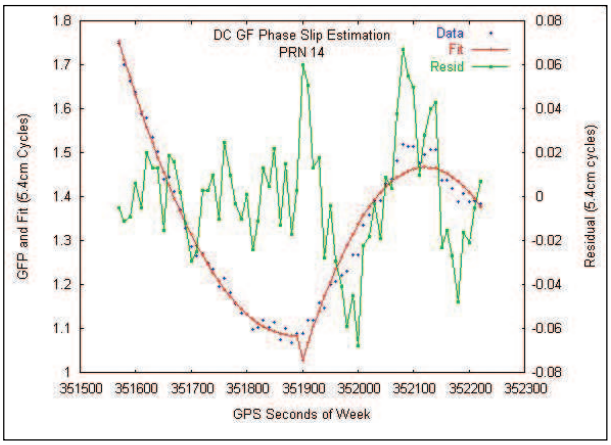
Ha magasabb szintre szeretnénk lépni, akkor közvetlenül a vevő által végzett megfigyelések eredményeivel kell dolgoznunk, ezt a lehetőséget viszont csak néhány nyílt forrású vagy szabadon elérhető program biztosítja. Az ***OpenSourceGPS*** például ***Zarlink*** lapkakészletre épülő ***GPS***-vevő kifejlesztését célozza. Az ***UNAVCO*** teqc eszközkészlete minőségbiztosítást végez, illetve a vevőtől érkező nyers adatokat feldolgozva ***RINEX*** formátumú üzeneteket állít elő, de zárt forrású. A ***GPSTk*** célja ezekkel szemben az, hogy a felhasználóknak a megfigyelések eredményeinek elérésén túl a feldolgozó algoritmusok továbbfejlesztésére is megadja a lehetőséget.

A GPS Toolkit

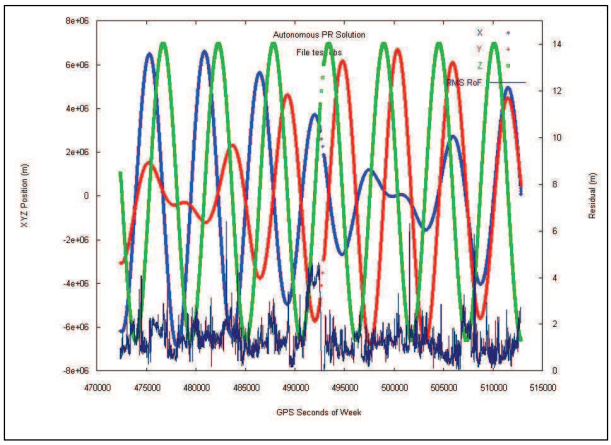
A ***GPS Toolkit (GPSTk)*** kódja tisztán ANSI C++ alapú. Géptípustól független, ***Linuxon***, ***Solarison*** és ***Microsoft Windowson*** egyaránt lefordítható és gond nélkül használható. Minden megtalálható benne, ami az önálló, konzolos programok írásához szükséges, illetve számos teljes alkalmazás is része. Tervezése nagymértékben objektumorientált. Minden a gpstk:: névtéren belül található. Egy ***RINEX*** formátumú megfigyelésfájlt például ilyen egyszerűen lehet olvasni és írni:

```
// RINEX fájl megnyitása, olvasása és írása
using namespace gpstk;
// bemeneti fájlfolyam
RinexObsStream rin(inputfile);
// kimeneti fájlfolyam
RinexObsStream rout(outputfile,
ios::out|ios::trunc);
DayTime nextTime; //dátum/idő objektum
RinexObsHeader head; //RINEX fejléc objektum
RinexObsData data; //RINEX adat objektum

// a RINEX fejléc beolvasása
rin >> head;
rout.header = rin.header;
rout << rout.header;

// végiglépkedés a különféle időpontokhoz tartozó
   ➥adatokon
while (rin >> data) {
nextTime = data.time;
// megfigyelési adat megváltoztatása
rout << data;
}
```

A könyvtár legfontosabb képességei a ***RINEX*** fájlok olvasására/írására épülnek. Része egy teljes értékű dátum- és időkezelő osztály, amellyel ***GPS*** és egyéb formátumú időcímkéket lehet manipulálni.
A ***RINEX*** fájlok kezelésén túl a ***GPSTk*** geodéziai koordináták (szélesség és hosszúság) kezelésére alkalmas, valamint ***GPS*** efemerisz számításokra használható osztályokat is tartalmaz. Ugyancsak található benne egy teljes értékű, sablon alapú mátrix- és vektorkezelő csomag, illetve ***GPS*** helymeghatározó és navigációs algoritmusok, melyeket nagy számú troposzféra modell egészít ki.
Végül, a terjesztés önálló programokat is tartalmaz. Találunk benne ***RINEX*** fájlok ellenőrzésére és módosítására, összegzé-



5. ábra A cikluscsúszás becsült mértéke



6. ábra Helyzetek interpolálása

© Kiskapu Kft. Minden jog fenntartva

sek készítésére, megfigyelési adatok eltávolítására vagy módosítására, a mondatok szakadásait javító, valamint a megszokott hibák és helyesbítő tényezők számítására alkalmas segédprogramot. Utóbbi például az ionoszférában a jel útvonalán található teljes elektrontartalmat képes meghatározni.

Az első lépések a GPS Toolkittel

A *GPS Toolkit* .tar állomány formájában tölthető le. (Lásd az internetes források részt.) Az eszközkészlet lefordításához szükség van a jam-re, a make egyik helyettesítőjére, illetve a forrásfájlok alapján leírásokat előállító ***Doxygenre***.

A fordítás menete nagyjából a következőképpen alakul:

```
tar xvzf gpstk-1.0.tar.gz
cd gpstk
jam
doxygen
su
jam -sPREFIX=/usr install
```

A fenti parancssorozattal lefordítjuk és – a /usr fa alá – telepítjük a ***GPSTk*** dinamikus és megosztott könyvtárait, vala-

© Kiskapu Kft. Minden jog fenntartva

mint a fejlécfájlokat. Létrejön egy doc alkönyvtár is, amely a ***GPSTk*** könyvtár HTML alapú leírásait tartalmazza.
Az alábbiakban három, az ARL:UT-n készített ***GPSTk*** példa-alkalmazást szeretnék ismertetni. A második ezek közül a ***GPSTk***-ban is szerepel.

Megnövelt pontosságú helymeghatározás

A ***GPSTk*** által előállított helyzetmegoldások jóval pontosabbak, a hibahatásokra érzéketlenebbek a ***GPS***-vevők által szolgáltatottaknál. A *2. ábrán* az ebből fakadó előnyöket szemléltettük; mindkét tengely a -10 – 10 méter tartományt ábrázolja.
Az A ábra a helyzetszámításokat, illetve keleti és északi irányú szórásukat szemlélteti. Egy fogyasztói szintű ***GPS***-vevő ilyen eredményeket szolgáltat. A B ábra a légköri késleltetések figyelembe vételével kapott helyzetbecsléseket szemlélteti. A közvetlen feldolgozás nemcsak a pontosságot, de a hibatűrést is javítja. A C ábrán láthatjuk, mi származik egy műhold meghibásodásából. A D ábrán követhetjük, hogy a hibás műhold ***GPSTk*** általi felismerésekor és kiiktatásakor mi történik.

Vivőfázis-szakadások helyesbítése

A ***GPS***-adatok feldolgozásakor sok gondot okoznak a vivőfázisban megjelenő szakadások. A fázisadatok felhasználása előtt fel kell ismerni és helyesbíteni kell a cikluscsúszásokat. A ***GPSTk*** csomag tartalmaz egy szakadásjavító alkalmazást, amely pontosan ezt a feladatot látja el. A javítási lehetőséget természetesen maga a könyvtár is biztosítja.
A ***GPSTk*** szakadásjavítója a kétfrekvenciás fázisadatok két lineáris kombinációját állítja elő, ezeket szélessávú fázistorzításnak és geometriamentes fázisnak nevezzük. Normál adatok esetére ezek alakulását a *3. ábra* szemlélteti. A szélessávú torzítás (vörös) zajos ugyan, de átlaga állandó. A geometriamentes fázis nem függ a vevő és a műhold geometriai viszonyától, de az ionoszféra miatt fellépő késleltetés erősen befolyásolja, pontosabban egyenesen arányos ezzel a késleltetéssel. Normál esetben az ionoszféra nyugodt és egyenletes, ám előfordul, hogy aktívabb, zavarosabb állapotba kerül, ilyenkor ez az érték széles tartományban változhat. A geometriamentes fázis és a szélessávú zaj az adatsor elején és végén egyaránt növekszik, a műhold ugyanis ekkor kel és nyugszik, ilyenkor a jelnek hosszabb útvonalat kell megtennie a légkörben.
A szakadásjavító először a szélessávú fázistorzításban keresi meg a csúszásokat. A *4. ábra* egy ilyen felismerésének esetét szemlélteti. Ha a szélessávú fázistorzításban csúszást talál, a kód a geometriamentes fázist is megvizsgálja, és abban is csúszást keres. A csúszás nagyságát úgy becsüli meg, hogy a csúszás két oldalán alacsony fokú polinomokkal közelíti az adatokat, extrapolál a csúszás pontjáig, majd különbséget számol.

Műholdak helyzetének interpolálása

A ***GPSTk*** egy másik az ARL:UT-n kidolgozott alkalmazásához egy alacsony pályán keringő, ***GPS***-vevővel rendelkező műholdra is szükség van. Ez a műhold a felette látható műholdak ***GPS***-adatait is begyűjti, ezeket felsőoldali adatoknak nevezzük, illetve az alatta láthatókét is, ezek lesznek az alsóoldali adatok. Az alsóoldali ***GPS***-jel meglehetősen hosszú utat tesz meg a légkörben, így kiválóan használható a légkör állapotának figyelésére. A felsőoldali adatok alapján végezhető el az alacsony pályán keringő, gyorsan mozgó műhold helyzetének meghatározása. A gond az, hogy a felsőoldali adatok gyűjtése kisebb – 10 másodperces – gyakorisággal történik, mint az alsóoldaliaké (1 másodperc), ám a nagyobb gyakorisággal érkező alsóoldali adatok feldolgozásához szükség van a műhold pontos helyzetére. Ennek a problémának a megoldására született egy ***GPSTk*** alapú program, amely beolvassa a ***GPS***-adatokat, a felsőoldali adatok alapján kiszámítja az alacsony pályájú műhold helyzetét, majd ezeket egy másodperces időosztásokra interpolálja. Az eredmény, az alacsony pályájú műhold Föld körüli útja a *6. ábrán* látható.

A GPSTk jövője

Nyílt forrású, ***GPS***-adatok feldolgozására alkalmas, a ***GPS Toolkithez*** hasonló szolgáltatásokat nyújtó csomag eddig még nem létezett – izgatottan várjuk, mi fejlődik majd ki belőle. A ***GPSTk*** széleskörű figyelemre számíthat. Az egyetemek a ***GPSTk***-t felhasználhatják a ***GPS***-adatok nyílt forrású kóddal való feldolgozására. A beágyazott eszközök fejlesztői olyan programokat állíthatnak össze, amelyek ***GPS*** alapú helymeghatározásra, valamint ***RINEX*** adatfájlok olvasására, írására és szerkesztésére képesek. A kutatók kiváló kiindulási alapnak találják majd tisztán program alapú ***GPS***-vevők megvalósításához.
Bár a ***GPSTk*** további sorsa elsősorban a felhasználók visszajelzéseitől és hozzájárulásától függ, a műholdas navigáció terén beálló változások is befolyásolni fogják. A közeljövőről annyit, hogy az első olyan műholdat, amely polgári használatra is ad majd kétfrekvenciás áltartományokat, várhatóan 2005-ben lövik fel. Az ***Európai Unió*** elindította a ***Galileot***, ennek célja a ***GPS***-sel kompatibilis, polgári irányítású szolgáltatás biztosítása – valószínűleg átrendezi majd az eddigi erőviszonyokat. Hosszútávon várható, hogy a ***GPS*** rendszer új jeleket fog tartalmazni az L5 és az M kódban. A ***GPSTk*** az alapszintű megfigyelési lehetőségekre építkezve megfelelő alapot kínál majd mindezen változások, újdonságok követéséhez és kihasználásához.
Bízunk abban, hogy az egyetemi hallgatók, a laboratóriumi kutatók, a rendszermérnökök és a programfejlesztők egyaránt ki tudják majd használni a ***GPS Toolkit*** által kínált lehetőségeket, és maguk is hozzájárulnak további fejlesztéséhez. Saját laborunkon belül is rengeteg előnnyel járt és jár a használata, ezért reméljük, hogy a ***GPSTk*** még számos újszerű ***GPS***-es alkalmazás elkészítésében játszhat szerepet.

Linux Journal 2004. szeptember, 125. szám

Dr. Brian W. Tolman kutató a Texasi Egyetem Alkalmazott Kutatások Laboratóriumában. Immár 18 éves tapasztalattal rendelkezik a GPS vonatkozású kutatásokban, adatelemzésekben és programfejlesztésekben. Doktori címét az austini Texasi Egyetemen, elméleti fizikából szerezte.

Ben Harris mérnökkutató az austini Texasi Egyetemen. Amikor éppen nem GPS-es témájú vizsgálatokat végez, doktori dolgozatával foglalkozik vagy csodálatos családjával van együtt, akkor robothalakat programoz a Pulp Fiction egyes jeleneteinek eljátszására.

© Kiskapu Kft. Minden jog fenntartva

Grafikus felület létrehozása a ps parancshoz a Mozilla segítségével

Készítsünk grafikus felületet parancssoros feladatainkhoz a Mozillával.

A ***Linux*** egyik nagyon előnyős tulajdonsága, hogy álnevek, parancsfájlok és egyéb trükkök segítségével könnyen hozhatunk létre új parancsokat. Ezek a módszerek valamennyien a parancssori felületre építenek. Vajon mi a helyzet akkor, ha grafikus felületre van szükségünk?

Kevés olyan eljárás létezik, amely nemcsak kifogástalan külsőt, de könnyű használhatóságot is nyújt. Ebben a cikkben egy olyan ígéretes lehetőséggel ismerkedhetünk meg, amely a ***Mozilla*** felületét használja. Egy meglehetősen bonyolult, de gyakori problémán keresztül vizsgáljuk meg a kérdést: hogyan jeleníthetjük meg minél szemléletesebben a `ps` parancs által szolgáltatott hierarchikus információkat. Ehhez szükségünk lesz a ***Mozilla*** egy friss (legalább 1.4-es) példányára.

Számos grafikus eszközkészletet használhatunk ***Linux*** alatt, kezdve az ***Xt***-től a ***Tcl/Tk***-ig. Ezeknek a készleteknek az oktatóanyaga rendszerint egy gomb létrehozásával kezdődik. S ha ez ennyire bevett szokás, próbáljuk ki mi is, majd haladjunk tovább. A Mozillában a grafikus felületek leírására az ***XML*** parancsformátuma használatos. Egy `button.xul` nevű, gombot meghatározó dokumentum a következőképpen fest:

```
<?xml version="1.0"?>
<window
➥xmlns="http://www.mozilla.org/keymaster/
➥gatekeeper/there.is.only.xul">
  <button label="Press Me"/>
</window>
```

A kicsit hosszúnak tűnő karakterlánc, a `http://www.mozilla.org/keymaster/gatekeeper/stb...` azt jelzi a ***Mozilla*** számára, hogy most nem egy közönséges ***HTML***-fájlról van szó, hanem ***XUL***-fájlról. A ***XUL*** az ***XML*** egy grafikus környezetek leírására használatos, ***Mozilla***-specifikus nyelvjárása. Jelenítsük meg a nyomógomb ablakát a következő paranccsal:

```
mozilla -chrome button.xul
```

Ez a példa nagyon egyszerű, nem is érdemes tovább foglalkoznunk vele, bár még egy egyszerű gomb viselkedésével kapcsolatban is sok minden elmondható lenne. A `ps` parancs kimenetének megjelenítése azonban ennél sokkal nagyratörőbb terv, úgyhogy inkább lépjünk tovább.

1. lista Parancssoros keret a ps számára

```
#!/bin/ksh
export COLUMNS=300
ps h -ew -o ´%p,%P,%C,%x,%z,%G,%n,%U,%a´
➥/tmp/psdata
```

A `<button>` eszközkészlet helyett próbáljuk ki a ***Mozilla*** és a ***XUL*** egyik komolyabb fegyverét, a `<tree>` eszközkészletet. Egy kis kódolásra is szükségünk lesz, no és jóval több ***XML***-re. Most a hangsúly inkább a gyors fejlesztésen van, nem cél a tökéletes program megalkotása. A kódolással kezdjük. Indulásképpen a `ps` elvégzi a kezdeti adatok begyűjtését. Az *1. listán* a 777 jogosultsági módú `psdata.ksh` fájlt látjuk. A kimenet minden minket érdeklő mezőt tartalmaz vesszőkkel elválasztva, a fejlécsor nélkül. A `PID` és a `PPID` kötelező részek, a maradék pedig olyan nem kötelező, de hasznos mezőkből áll, mint például a `COMMAND`. Ez minden, amit a hagyományos ***Linux*** szükségessé tesz.

A kód többi része a ***Mozillától*** függ. Az előre lefordított rendszercsomagok legalább két futtatható fájlt tartalmaznak, a `mozilla`-t, és a `regxpcom`-ot. Mi az `xpcshell` bináris állományt is használni fogjuk. Ez az állománya ***Mozillában*** a Perl parancsértelmező ***JavaScript*** megfelelője és nem rendelkezik grafikus támogatással. Az `xpcshell` olykor a fejlesztés jó kiindulópontja lehet, de soha nem nélkülözhetetlen. Ennek megszerzéséhez a ***Mozilla*** egy teljes lefordított változatára van szükségünk. Ellenőrizzük először is a `www.mozilla.org/build` oldalon az eszközlánccal (toolchain) kapcsolatos előfeltételeket. Ezután töltsük le a forrást az FTP vagy távoli CVS segítségével. Inkább egy fő kiadást, mint a naponta változó legfrissebb változatot ajánlom. A kicsomagolás után kövessük a szabványos fordítási lépéseket:

```
cd mozilla
./configure –disable-debug
```

© Kiskapu Kft. Minden jog fenntartva

2. lista Az idegen adatok kötegelt betöltése az xpcshell-be

```
const Cc = Components.classes;
const Ci = Components.interfaces;

var klass = {};
var psdata = null; // last results from ps(1).

klass.file = Cc["@mozilla.org/file/local;1"];
klass.process =
      Cc["@mozilla.org/process/util;1"];
klass.stream
     = Cc["@mozilla.org/network/file-input-
     ➥stream;1"];
klass.jsstream
     = Cc["@mozilla.org/scriptableinputstream;1"];

function execute_ps() {
  // freeze until ps(1) is finished.
  var blocking = true, argv = [], result = {};
  var path =
      "/home/nrm/writing/psviewer/psdata.ksh"

  var file
   = klass.file.createInstance(Ci.nsILocalFile);
  var process
   = klass.process.createInstance(Ci.nsIProcess);

  file.initWithPath(path);
  process.init(file);
  process.run(blocking, argv,argv.length,
        ➥result);
}

function read_raw_data() {
  const path = "/tmp/psdata";
  var mode_mask = 0x01, perm_mask = 0;  //open(2)

  var file
   = klass.file.createInstance(Ci.nsILocalFile);
  file.initWithPath(path);

  var stream = klass.stream.createInstance(
      ➥Ci.nsIFileInputStream);

  stream.init(file, mode_mask, perm_mask, 0);

  var jsstream = klass.jsstream.createInstance(
      ➥Ci.nsIScriptableInputStream);

  jsstream.init(stream);

  var data = jsstream.read(file.fileSize);

  // got the file content. break it down.

  data = data.split("\012");

  for (var i = 0; i < data.length; i++)
  {
    data[i] = data[i].replace(/\s*,\s*/,",");
    data[i] = data[i].replace(/^\s*/,"");
    psdata.push(data[i].split(","));
  }
}

execute_ps();
read_raw_data();
```

```
make
make install
```

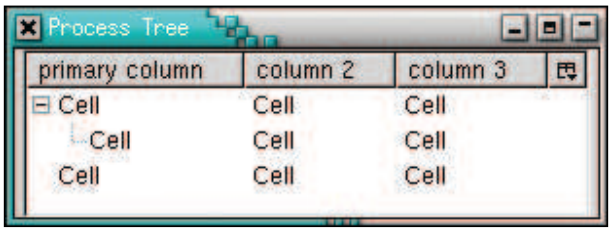
A hibakeresésre használt változatok lassúak és tele vannak hibakereső részletekkel. Igaz ugyan, hogy ezek ártalmatlanok, de most inkább kerüljük őket. Az ilyen kódnak a fordítása is tovább tart egy órával, és akár 1 GB helyet is elfoglalhat.
A kapott bináris állományokat a `mozilla/dist/bin` könyvtárban találjuk majd. Ezeket futtathatjuk ebből a könyvtárból, vagy bármilyen más helyről, amennyiben a `MOZILLA_FIVE_HOME` és `LD_LIBRARY_PATH` változók értéke be van állítva. Most már minden szükséges bináris állomány és héjprogram elérhető.
A *Perl* esetén a `ps` kimenetét valahogy el kell juttatni a programozási környezetbe. Ebben az esetben az eszköz egy *JavaScript* parancsértelmező. Ennek a végrehajtásához nem elég egy nyelvi parancsformátum, az I/O támogatására is szükségünk van. A *Perl*-ben a támogatás a nyelvi függvények szintjén került megvalósításra, ezzel szemben a *JavaScript* nem rendelkezik I/O-függvényekkel.

A *Mozillában* ez az I/O támogatás objektumok használatával érhető el, amelyeket azonban nem szolgáltathat egy könyvtár, mivel a nyelvi alap nem rendelkezik I/O-függvényekkel. Vagyis a *Perl* `use` vagy `require` parancsa ebben az esetben nem fog működni. Olyan áttételes műveleteket sem alkalmazhatunk, mint például az `echo 'pwd'`. Ehelyett a *Mozilla* az *XPCOM*-ot kínálja.
Az *XPCOM* a *Microsoft*-féle *COM* (Component Object Modell, komponens objektum-modell) egy megvalósítása, és használható *Linux*/*UNIX*, *Windows* és *Macintosh* rendszereken egyaránt. Jelenleg a működése egyetlen folyamatra korlátozódik, vagyis nincs lehetőség *DCOM* (Distributed Component Object Modell, elosztott komponens objektum-modell) használatára. Az *XPCOM*/*COM* nyújtja a leggyorsabb megoldást arra, hogy egy parancsnyelvi környezetet új lehetőségekkel bővítsünk. Működése közben egy előre lefordított objektumot (például *C* vagy *C++* objektumot) kapcsol össze a parancsnyelv objektumhivatkozásával.
A legközelebbi *Perl*-megfelelő az *XM*, de míg az *XPCOM* nem igényli a program újrafűzését, az *XM* igen.

3. lista Tiszta XUL kód a <tree> eszköz statikus tartalommal történő megvalósítására

```
<?xml version="1.0"?>
<?xml-stylesheet
  href="chrome://global/skin/global.css"
  type="text/css"?>
<!DOCTYPE window>
<window xmlns="http://www.mozilla.org/keymaster/
            ➥gatekeeper/there.is.only.xul"
            ➥title="Process Tree">

<tree id="t1" flex="1">
  <treecols>
    <treecol flex="1" id="A"
     ➥label="primary column" primary="true"/>
    <treecol flex="1" id="B" label="column 2"/>
    <treecol flex="1" id="B" label="column 3"/>
  </treecols>

  <treechildren id="titems" flex="1">

    <treeitem id="row1" container="true"
                ➥open="true">
      <treerow>
        <treecell label="Cell"/>
        <treecell label="Cell"/>
        <treecell label="Cell"/>
      </treerow>

      <treechildren>
        <treeitem>
          <treerow>
            <treecell label="Cell"/>
            <treecell label="Cell"/>
            <treecell label="Cell"/>
          </treerow>
        </treeitem>
      </treechildren>
    </treeitem>

    <treeitem>
      <treerow>
        <treecell label="Cell"/>
        <treecell label="Cell"/>
        <treecell label="Cell"/>
      </treerow>
    </treeitem>

 </treechildren>
</tree>

</window>
```

Bár a *Mozilla* alapértelmezésben is több ezer *XPCOM* objektumot tartalmaz, az *XPCOM* mégsem egy *Java*-szerű virtuális gép, hanem rendszerint olyan lefordított kód, amely hatékonyságát a hardverközeli futás biztosítja.

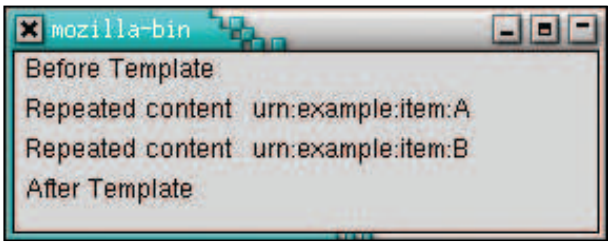


1. kép Egyszerű <tree> eszköz startikus XUL tartalommal

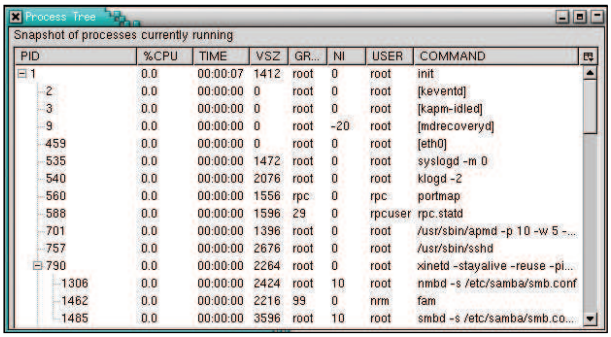
Furcsának tűnhet, hogy *Linuxon* a *Microsoft* elgondolását alkalmazzuk, de az *XPCOM* teljesen nyílt forráskódú és a *UNIX* terén olyan szerepet tölt be, amelyre sokáig nem volt példa: a *Linux*/*UNIX* rendszereken egyszerűen nem létezett ilyen jól használható, közepes méretű komponensmodell. A múltban is volt *CORBA*, és voltak a dinamikus könyvtárak (dll), de az egyik túlságosan nehézsúlyú, a másik túlzottan pehelysúlyú megoldást jelentett. Az *XPCOM* ezzel szemben tökéletesen illeszkedik a közepes méretű feladatokhoz. Rendkívül jól használható azokon a területeken, ahol az alkalmazás viszonylag nagy bináris állományokból áll és a megvalósítás alapvetően teljesítményközpontú.
Az *XPCOM* illetve *COM* használatára a *Windows* `QueryInterface()` metódusának gyakori hívása a jellemző, de most a *Linux*-programozók érzékenysége miatt a cikkben helyette inkább a `createInstance()` és `getService()` metódusokat fogjuk használni. Azért jó tudni, hogy a `QueryInterface()` is rendelkezésre áll.
Térjünk vissza a kódra. Olvassuk be a `ps` kimenetét a *2. listán* látható módon.
A lista első része beállít néhány globális változót.
A `Components` egy már létező objektum, amely valamennyi komponensnek nevezett *XPCOM*-objektum és az általuk támogatott (*Java* vagy *COM* értelemben vett) felületek könyvtára. Egy *XPCOM*-objektum megszerzéséhez keressük meg a megfelelő komponenst (a nevét a `Contract ID` nevű karakterlánc tárolja) és hozzunk létre hozzá egy megjelenési (interface) objektumot (amely szintén egy karakterlánc vagy tulajdonság neve alapján kaphatja a nevét – mi itt az utóbbit használjuk). A komponensek újbóli felhasználása elterjedt gyakorlat, így ha egyszer létrehoztuk, akkor a `klass` objektum egy alkalmas jellemzőjeként kerülnek mentésre – a `class` a *JavaScript* egy fenntartott kulcsszava.
A megadott függvények foglalják el a listában a fennmaradó helyeket. Az `execute_ps()` egyszerűen egy újabb folyamatot futtat: a `ps` keretének parancsfájlját. Ehhez szüksége van egy fájl objektumra (mégpedig egy `nsILocalFile` típusúra) és egy `process` (folyamat) objektumra (`nsIProcess` típus). A `run()` a `fork()` használatával hívja meg a folyamatot. A *Mozillát* úgy fejlesztették, hogy mindezt hordozható formában tegye, de esetünkben most csak a Linux támogatott, mert a futtatható fájl elérési útja konstansként szerepel a kódban. A másik függvény, a `read_raw_data()` az adatokat olvassa be. A *Mozilla* az adatfolyam, átvitel és csatorna elveket használja, vagyis ugyanazokat, amelyeket a *Java* néhány magas szintű szolgáltatása, de mindezt az osztályok létrehozásának bonyodalma nélkül. Nekünk egy fáljobjektumra van szükségünk a `ps`-től jövő adatok tárolá-

© Kiskapu Kft. Minden jog fenntartva

© Kiskapu Kft. Minden jog fenntartva



2. kép Statikus RDF-tartalomra épülő egyszerű sablonnal támogatott grafikus felület



3. kép A végső <tree> eszköz, amelynek RDF-adatait a JavaScriptből merítettük

sára. A `stream` objektum egy utat nyit meg ennek a tartalomnak a fájl felé. Egy kis trükkre is szükség van: az eredeti `stream` objektumot egy másik különleges `stream`-objektumba kell ágyazni, amelyre parancsok adhatók ki. Egyetlen `read()` hívással a fájl egész tartalmát egy karakterláncban helyezzük el. A következő lépésben néhány ***Perl***-szerű szabályos kifejezés boszorkányos ügyességgel először sorok tömbjeire majd egymásba ágyazott tömbökké alakítja a karakterlánc tartalmát. Minden adatot karakterként kezelünk. Az adatok feldolgozásának helyességét az `xpcshell` által biztosított elemi `print()` paranccsal ellenőrizhetjük. Sajnos a ***Mozilla*** jelenleg nem támogatja a PID azonosítók visszakeresését, így `/tmp/psdata.$$` jellegű fájlokat még nem használhatunk. Ugyanakkor valószínű, hogy ennek a megoldása sem fog sokáig váratni magára.

Ebben a parancsfájlban egy sereg ***XPCOM***-objektumot találhatunk, de vajon hogyan tudjuk a megfelelőket megkeresni? Ahogy bármelyik programozói könyvtár esetén, itt is rendelkezésünkre áll egy referenciaanyag. Keressük meg a ***Mozilla*** forráskódjának .IDL kiterjesztésű fájljait (ezek a `mozilla/dist/idl` könyvtárban helyezkednek el), nézzünk körül a ***Világhálón***, vagy olvassunk el egy könyvet a témában.

Kezdetnek ennyit a parancsfájlokról, a parancsfájlok írása és a táblázatos adatok témája a ***Linux*** amúgy is jól ismert területe. A grafikus felület létrehozásához a ***Mozillának*** az ***XML***-re ezen belül is a ***XUL***-ra van szüksége. Ez a parancssortól nagymértékben eltérő terület és ahhoz, hogy boldoguljunk vele, közelebbről meg kell ismernünk. A folyamatot éppen ezért most könnyen érthető lépésekre osztottam fel. Pillantsunk először is a *3. lista* és az *1. kép* által mutatott ***XUL*** `<tree>` eszközre. A fa megjelenése igen tetszetős, mivel a `<?xml-slylesheet?>` feldolgozási utasítás feltétel nélkül

4. lista Egyszerű tartalomtól függő sablon megvalósítása XUL-kóddal

```
<?xml version="1.0"?>
<?xml-stylesheet
  href="chrome://global/skin/"
  type="text/css"?>
<!DOCTYPE window>
<window xmlns="http://www.mozilla.org/keymaster/
            ➥gatekeeper/there.is.only.xul">
 <description value="Before Template"/>
 <vbox datasources="trivial.rdf"
  ➥ref="urn:example:items"

➥containment="http://www.example.org/
➥TestData#items">

  <template>
   <rule>
    <conditions>
     <content uri="?uri"/>
     <member container="?uri" child="?note"/>
    </conditions>
    <action>
     <hbox uri="?note">
      <description value="Repeated content"/>
      <description value="?note"/>
     </hbox>
    </action>
   </rule>
  </template>
 </vbox>
 <description value="After Template"/>
</window>
```

5. lista A 4. lista sablonjához illő trivial.rdf fájl

```
<?xml version="1.0"?>
<RDF
 xmlns:TD="http://www.example.org/TestData#"
 xmlns="http://www.w3.org/1999/02/ 22-rdf-
      ➥syntax-ns#">

  <Description about="urn:example:root">
    <T:items>
      <Seq about="urn:example:items">
        <li resource="urn:example:item:A"/>
        <li resource="urn:example:item:B"/>
      </Seq>
    </T:items>

  </Description>
</RDF>
```

6. lista A ps-adatok faszerkezet-nézetének végső XUL-kódja

```
<?xml version="1.0"?>
<?xml-stylesheet
 href="chrome://global/skin/global.css"
 type="text/css"?>
<!DOCTYPE window>
<window xmlns="http://www.mozilla.org/keymaster/
        ➥gatekeeper/there.is.only.xul"
        ➥title="Process Tree" flex="1">
 <script src="tree.js"/>

 <vbox flex="1">
  <description>
   Snapshot of processes currently running
  </description>

 <tree id="proc-tree"
  ➥flex="1"
  ➥datasources="rdf:null"
  ➥ref="http://www.example.org/ProcData#ProcList"
  ➥containment="http://www.example.org/
  ➥ProcData#child">

  <treecols>
   <treecol id="pid" primary="true" label="PID"
            ➥minwidth="75"/>
   <splitter class="tree-splitter"/>
   <treecol id="pcpu" label="%CPU"
            ➥minwidth="40"/>
   <splitter class="tree-splitter"/>
   <treecol id="time" label="TIME"
            ➥minwidth="40"/>
   <splitter class="tree-splitter"/>
   <treecol id="vsz" label="VSZ" minwidth="40"/>
   <splitter class="tree-splitter"/>
   <treecol id="group" label="GROUP"
            ➥minwidth="40"/>
   <splitter class="tree-splitter"/>
   <treecol id="nice" label="NI" minwidth="40"/>
   <splitter class="tree-splitter"/>
   <treecol id="user" label="USER"
            ➥minwidth="40"/>
   <splitter class="tree-splitter"/>
   <treecol flex="1" id="args" label="COMMAND"
            ➥minwidth="40"/>
  </treecols>
  <template>
   <treechildren>
    <treeitem open="true" uri="rdf:*">
     <treerow>
      <treecell label=
        ➥"rdf:http://www.example.org/
        ➥ProcData#pid"/>
      <treecell label=
        ➥"rdf:http://www.example.org/
        ➥ProcData#pcpu"/>
      <treecell label=
        ➥"rdf:http://www.example.org/
        ➥ProcData#time"/>
      <treecell label=
        ➥"rdf:http://www.example.org/
        ➥ProcData#vsz"/>
      <treecell label=
        ➥"rdf:http://www.example.org/
        ➥ProcData#group"/>
      <treecell label=
        ➥"rdf:http://www.example.org/
        ➥ProcData#nice"/>
      <treecell label=
        ➥"rdf:http://www.example.org/
        ➥ProcData#user"/>
      <treecell label=
        ➥"rdf:http://www.example.org/
        ➥ProcData#args"/>
     </treerow>
    </treeitem>
   </treechildren>
  </template>
 </tree>
</vbox>

</window>
```

lemásolja az aktuális *Mozilla*-témát. Ha el szeretnénk hagyni az alapvető vezérlőgombokat és egyéb díszeket, a *Mozilla* futtatható fájlját használhatjuk a `-chrome` kapcsolóval:

```
mozilla -chrome static_tree.xul
```

Az *XML*-tartalom (amire ezentúl kódként hivatkozom), egy kicsit a *HTML* `<table>` címkéjére hasonlít: megadtuk mind az oszlopfejlécek mind pedig az adatsorok tartalmát.
A trükk a `<treeitem>` címkében rejlik, amely tartalmazhat egy `<treechildren>` címkét, amely lehetővé teszi, hogy ne csak egylevelű csomópontokat, hanem további faágakat is elhelyezhessünk a fán. Ahogy az 1. képen is látszik, a faeszköz számos interaktív szolgáltatást is kínál: az ágak ugyanúgy nyithatók/zárhatók, ahogy egy *Nautilus* vagy *Windows Explorer* fájlkezelő programban, oszlopokat hozhatunk létre, vagy törölhetünk az oszlopneveket tartalmazó fejléc jobb szélén lévő oszloprendező gombbal.
Ha akarnánk, tulajdonképpen használhatnánk *JavaScript* parancsfájlokat is a `ps` adatainak ebbe a *XUL*-dokumentumba való dinamikus betöltéséhez. Ez nem is lenne nehéz, és az összes *W3C DOM* csatolófelület elérhető a megvalósításhoz. Kezdjük az `Element`-objektumok hozzáadásával, vagy használjuk inkább az `innerHTML` jellemzőt. Ezzel a cikkel azonban szeretném egy kicsit magasabbra tenni a lécet, és inkább egy teljesen adatvezérelt megközelítést mutatok be, amelyben nem kell egyetlen fát sem kézzel létrehoznunk.

© Kiskapu Kft. Minden jog fenntartva

© Kiskapu Kft. Minden jog fenntartva

7. lista Az adatok parancsfájlból RDF-adatforrásba történő átvitele.

```
 // --- globals ---
klass.datasource
 = Cc["@mozilla.org/rdf/datasource;1" +
      "?name=in-memory-datasource"];
klass.rdf
 = Cc["@mozilla.org/rdf/rdf-service;1"];

var schema = "http://www.example.org/ProcData#";
var props =
  [ "pid", "ppid", "pcpu", "time", "vsz",
    "group", "nice", "user", "args" ];

var rdf = klass.rdf.getService(Ci.nsIRDFService);

var root  = rdf.GetResource(schema + "ProcList");
var child = rdf.GetResource(schema + "child");
var preds = [];

for (var p in props)
  preds[p] = rdf.GetResource(schema + props[p]);

// --- mainline ---

window.addEventListener("load",load_handler,true);

// --- functions ---

function update_tree() {
  var tree = document.getElementById
           ➥("proc-tree");

  // get the in-memory ds, not the
rdf:localstore
  var ds   = tree.database.GetDataSources();
      ds   = ( ds.getNext(), ds.getNext() );
      ds   =
       ➥ds.QueryInterface(Ci.nsIRDFDataSource);

  var sub, pred, obj;

  for (var i=0; i < psdata.length; i++)
  {
    if ( psdata[i][1] == "0" ) // a root node
      sub = root;
    else                       // a child node
      sub = rdf.GetResource(
              ➥schema + "process-" +
              ➥psdata[i][1]);

    pred = child;
    obj = rdf.GetResource(
              ➥schema + "process-" +
              ➥psdata[i][0]);

    ds.Assert(sub, pred, obj, true);

    // add all properties for this process

    sub = obj;
    for (var j=0; j < psdata[i].length; j++)
    {
      pred = preds[j];
      obj = rdf.GetLiteral(psdata[i][j]);
      ds.Assert(sub, pred, obj, true);
    }
  }
}

function load_handler() {
  var tree = document.getElementById("proc-
           ➥tree");
  var ds = klass.datasource.createInstance(
          ➥Ci.nsIRDFInMemoryDataSource);
  tree.database.AddDataSource(ds);

  update_tree();
}
```

A *4. lista* és a *2. kép* egy fa nélküli ***XUL*** grafikus felületet mutat. Ebben egy `<template>` címkét alkalmaztunk a cél eléréséhez.

A ***XUL***-sablon nem a ***C++*** sablonokhoz, hanem inkább egy jelentésmintához hasonlítható, amely az ismétlődő adatcsoportok alapjául szolgál. A sablon egy `<vbox>` címkével kezdődik, amely rendelkezik egy `datasources=` tulajdonsággal. A `<template>` szakasz `<action>` része tartalmazza azokat a soronként ismétlődő adattartalmakat, amelyeket a `<conditions>` rész azonosít a ***trivial.rdf*** fájlban. Ha közepes szintű tudással rendelkezünk a `make` vagy ***SQL*** terén, esetleg van némi ismeretünk a ***Lisp***, ***Scheme*** vagy ***Prolog*** nyelvekkel kapcsolatban, nem okozhat gondot a rendszer működésének megértése. Az *5. listán* látjuk a 2. képhez tartozó ***trivial.rdf*** fájl tartalmát.

Ha módosítjuk ezt a fájlt, a 2. kép akkor is megváltozhat, ha a 4. lista közben nem módosult. Ez tehát egy adatvezérelt megoldás. Ez a fájl ***RDF***-típusú, amely a ***W3C*** szigorúbb szabványai közé tartozik. Az alkalmazott logikai szerkezet lényegében egy csomópontokból álló gráf, amelyben minden csomópont három adategységet hordoz. Az egyes egységek nevei `subject` (téma), `predicate` vagy `property` (állítás, jellemző) és `object` (céltárgy). Az egyszerű gráfok fák, így az 5. lista is egy fát ír le. Ha összekapcsoljuk a 4. lista `<hbox>` címkéjét az 5. listában lévő `<li>` címkékkel, a 2. képen látható eredményt kapjuk. Ez valami olyasmi, mint az ***SQL*** táblakapcsolata vagy a `join` parancs. Egyelőre annyit jegyezzünk meg, hogy a 4. lista `ref=` tulajdonsága az 5. lista `<Seq>` címkéjének felel meg. A kettőt így kapcsolja össze a ***Mozilla*** sablonfeldolgozó logikája. A ***Mozilla*** RDF-

támogatása nem túl szigorú, így majdnem az összes URI és URL helyben feldolgozható, mintha változóról vagy konstansról lenne szó. Ebben a cikkben is végig kihasználtuk ezt a tulajdonságot. Írjunk be az 5. listába még egy `<li>` címkét, indítsuk újra a *Mozilla*-t és jelenítsük meg ismételten az oldalt.

A faszerkezet jó megoldást nyújt a folyamatok hierarchikus listájának megjelenítésére és a `<template>` is megfelelő eszköz a fa megjelenésének adatokból történő vezérlésére. Bár nincs olyan RDF-dokumentumunk, amit használni tudnánk, ehelyett rendelkezünk a rekordok egy *JavaScrtipt*-tömbjével. A megoldás az, hogy összekapcsoljuk a `<tree>` és a `<template>` címkéket, és az RDF-fájlnak az `rdf:null` értéket adjuk, ami azt jelenti, hogy nincs ilyen fájl megadva. Egy parancsfájlt használunk arra, hogy az RDF-tartalmat közvetlenül az adatokból állítsuk elő. Az RDF különleges felépítésének köszönhetően a tartalom minden gond nélkül áttölthető a sablonba, és így a működés egészen egyszerű. Ez egy sokkal tisztább, bár kétségkívül kényesebb megoldás, mint a *XUL*-fa kézzel történő felépítése a *JavaScript*-ből. Az RDF és a sablonok használatának egy másik előnyös tulajdonsága, hogy a fa bármikor egyszerűen frissíthető. Ez azt jelenti, hogy az ablakban dinamikusan megjeleníthetőek a ps(1) parancsból származó adatok, mintha a `watch ps H` egy grafikus megfelelője futna. Ennek bemutatása azonban már meghaladja e cikk kereteit.

A `<tree>` és `<template>` címkék együttes használatával létrehozott XUL-dokumentum eredménye a *3. képen* és a *6. listán* látható. Ebben is megtalálhatjuk a `datasource=` és `ref=` tulajdonságokat, valamint a `<template>` címkét. Az URL-ek az `rdf:` karakterlánccal kezdődnek és azokat a pontokat mutatják, ahol a sablonokba RDF-adatokat kell beillesztenünk. A korábbi példában a változók kérdőjellel kezdődtek, ezek megadására kétféle parancsformátum áll rendelkezésre, és természetesen minden sorhoz és minden oszlophoz egy ilyen adat tartozik.

A `<splitter>` címke egyszerűen egy felhasználóbarát kiegészítés, amely lehetővé teszi az oszlopok átméretezését. Ez javítja az olvashatóságot, ahogy a `minwith=` és `flex=` attribútumok is. A 3. képen látható, ahogy a folyamatok hierarchikusan megjelenített rendszere természetes módon kitölti a faszerkezetet.

A 6. lista felső részén lévő `<script>` címke beemeli a 2. lista teljes kódját egy kis kiegészítéssel. Az ilyen parancsrészek beemelésekor rögtön adódik egy biztonsági probléma mégpedig az, hogy a *Mozilla* eljárásának kell biztosítania a távoli adatok és parancsfájlok biztonságos megjelenítését, éppen úgy, mint a *HTML*-oldalakét. Ez olyan, mint a *Java*-kiszolgáló eredet-szabálya. Az `xpcshell` nélkülöz minden biztonsági beállítást, a *Mozilla* bináris állománya azonban rendelkezik a szokásos biztonsággal. A beállítások alapos átdolgozásával felülkerekedhetünk ezeken a megszorításokon, de egyszerűbb a parancsfájlt egy csomagként regisztrálni. Ehhez minden fájlt át kell helyeznünk a Mozilla telepítőjének `chrome` könyvtárába, ahol ezek a biztonsági korlátok feloldódnak. Ennek módjáról rövidesen szó lesz, de először befejezzük a programunkat egy olyan parancsfájllal, ami a `ps` adatait az egyszerű

© Kiskapu Kft. Minden jog fenntartva

© Kiskapu Kft. Minden jog fenntartva

8. lista A psviewer csomag chrome-regisztrációjához szükséges információk.

```
<?xml version="1.0"?>
<RDF xmlns=
"http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:chrome="http://www.mozilla.org/rdf/chrome#
">

 <Seq about="urn:mozilla:package:root">
  <li resource="urn:mozilla:package:psviewer"/>
 </Seq>

 <Description
➥about="urn:mozilla:package:psviewer"
        ➥chrome:displayName="PSViewer"
        ➥chrome:author="Nigel McFarlane"
        ➥chrome:name="psviewer">
 </Description>
</RDF>
```

JavaScript adatszerkezetből egy RDF-adatforrásba mozgatja át. Ez a parancsfájl fogja helyettesíteni a korábban használt statikus RDF-fájlt (lásd a *7. listát*).
A 7. listán azt a parancsfájlt láthatjuk, amely a statikus RDF-fájlt fogja helyettesíteni. A faszerkezet sablonja által használt RDF-fájlhoz egy többlépcsős folyamat során tudunk *JavaScript* adatokat adni. A *Mozilla* egy adatforrásnak (datasource) nevezett objektumba olvassa be az RDF-adatokat. Mivel korábban az `rdf:null` meghatározást használtuk, nem létezik adatforrás objektumunk, ezért létre kell hoznunk egyet és hozzákapcsolnunk a sablonhoz. Ezt a dokumentum biztonságos betöltése után a `load_handler()` eljárás végzi el. A betöltés közben működő kezelőeljárás egy megszokott *HTML*-trükk, ami ugyanilyen jól alkalmazható a *XUL* esetében is. Ezt az adatforrást ezután az `update_tree()` függvény tölti fel az RDF-tartalommal a sablon számára. Ez igen egyszerűen történik, két egymásba ágyazott ciklus lépked végig a *JavaScript*-tömb adatain. A `ps(1)` minden egyes folyamatánál meghívódik az `Assert()` függvény, amely létrehoz egy RDF adatcsomópontot (három elemből álló adathármast), amely meghatározza, hogy a `PPID X` rendelkezik egy `PID Y` gyerekkel és további RDF-csomópontokat, amelyek szerint a `PID X`-hez a `USER` A tartozik vagy a `GROUP B`. A `<template>` és a `<tree>` címkék közösen dolgoznak azon, hogy ezek az RDF-csomópontok önműködően faszerkezetbe rendeződjenek, éppen úgy, ahogy a `make` értékeli ki egy adott `Makefile`-ban lévő összes fájl függőségi-fáját. Ezzel a statikus RDF-fájl helyére lépő parancsfájllal el is készültünk az egyszerű folyamatfigyelő programunkkal. Végül az alábbi lépéseket kell tennünk a biztonsági intézkedések feloldásához szükséges kódregisztráció érdekében:

```
M5H = $MOZILLA_FIVE_HOME
mkdir -p $M5H/chrome/psviewer/content
cp *   $M5H/chrome/psviewer/content
vi     $M5H/chrome/psviewer/content/contents.rdf
vi     $M5H/chrome/installed-chrome.txt
```

A `vi` első futtatásával létrehozzuk a `contents.rdf` fájlt. Ennek pontosan a *8. listán* látható kódot kell tartalmaznia. A következő lépésben hozzáadjuk ehhez a fájlhoz az ***installed-chrome.txt*** fájl tartalmát, amely egyetlen sor:

```
content,install,url,resource:/chrome/psviewer/
➥content/
```

A *Mozilla* az elindításkor megvizsgálja ez utóbbi fájlt, és ha változást észlel benne, akkor a felsorolt könyvtárakban megkeresi a ***contents.rdf*** fájlokat. Ezeket beolvassa, és a `make`-hez hasonlóan megjegyzi az összes létező csomagot. Minden ismert csomag teljes biztonsági hozzáféréssel rendelkezik, a 8. lista pedig hozzáadja ezekhez a `psviewer` csomagot is. A biztonságos fájlok most már egyetlen URL segítségével megjeleníthetők és futtathatók:

```
mozilla -chrome
chrome://psviewer/content/tree.xul
```

ahelyett, hogy azt kellene írnunk, hogy:

```
mozilla -chrome
file:///home/nrm/psviewer/tree.xul
```

A `pswiewer` a *Mozilla* telepítőjén belül első osztályú helyet foglal el. Ha szükséges, más programokkal is összeépíthetjük, így a *Firefox*/*Firebird* böngészővel vagy a *Thunderbird* levelezőügyféllel. Hozzáadhatjuk a Tools (Eszközök) menühöz is új menüpontként.
Bár nagyon sok módszertani lehetőséget érintettünk ebben a cikkben, nem szabad elkövetnünk azt a hibát, hogy rögtön mindet alkalmazni akarjuk. Mivel az *XML* használata a *Mozillában* nem túl jól dokumentált dolog, könnyen belebonyolódhatunk. Jobban jár az, aki valamilyen egyszerű feladattal kezdi, és csak fokozatosan közelít az itt bemutatott érdekes megoldások felé. Habár a `ps` kimenete dinamikus *HTML*-oldalként is megjeleníthető lenne, a *XUL* végső soron nagyobb teljesítményű és szebb, a munkaasztalba teljesen beilleszthető grafikus megjelenítést tesz lehetővé.
A *Mozilla* egy még felfedezésre váró hatékony grafikus környezet. Komoly az esélye, hogy *Linux* alatt olyan szerepet töltsön be, mint amilyet a *Visual Basic* a *Windows* alatt. Mi több, a *Mozilla* hordozható, operációs rendszertől független megoldásokat tesz lehetővé. Az elkészült alkalmazásunk működhet *BSD*, *HP-UX*, *SunOS*, *AIX* és *Mac OS X* operációs rendszereken is a *Linux* mellett.

Linux Journal 2004. július, 123. szám

Nigel McFarlane kiterjedt programozói tapasztalattal rendelkező, a tudomány és technika területén tevékenykedő szabadúszó író. Legfrissebb könyve a Rapid Application Developement with Mozilla (Gyors alkalmazásfejlesztés a Mozilla segítségével), ISBN 0131423436.
Az nrm@kingtide.com.au címen léphetünk vele kapcsolatba.

© Kiskapu Kft. Minden jog fenntartva

Kódolatlan szövegekkel dolgozó alkalmazások feltámasztása az Stunnel segítségével

Régi alkalmazásainkat módosításuk nélkül is párosíthatjuk korszerű titkosítási megoldásokkal. Mick Bauer elmondja, hogy az SSL megjelenése előtti programokat hogyan tudjuk naprakésszé varázsolni.

A világon rengeteg a munkáját jól végző, az idők próbáját kiállt, ám biztonsági szempontból rémálomnak számító hálózati alkalmazás létezik. ***Telnet***? Lenyűgöző egyszerűség és sokoldalúság, ám a bejelentkezési adatokat nyílt szövegben továbbítja. ***rcp***? Ismert, kiváló parancsfájlok írhatók hozzá, ám az ***rhosts*** alapú hitelesítés ideje, köszönhetően az IP-címek hamisításának, lejárt. Természetesen megtehetjük, hogy megszokott igavonóinkat titkosított utódaikra cseréljük le – ekkor az ***SSH*** felváltja a ***telnetet***, az ***scp*** vagy az ***SSH*** feletti ***rsync*** pedig az ***rcp***-t. Van azonban más lehetőségünk is, mégpedig az, hogy általános célú ***IPSec*** alagutat építünk ki minden olyan távoli állomás felé, amellyel adatcserét szeretnénk folytatni. Az utóbbi megoldás nyilván túlzás, az előbbit viszont sokszor könnyebb elképzelni, mint megvalósítani, különösen, ha az adott környezetben használt alkalmazások köre nem a mi ellenőrzésünk alá esik. Szerencsére arra is van lehetőség, hogy a hagyományos hálózati alkalmazásokat erőteljes titkosítással bővítsük.

Ez a lehetőség a nagy tudású ***SSL***-burkoló, az ***Stunnel*** használata. Ez alkalommal szeretném elmondani, hogy az ***Stunnel 4.0*** és az ***OpenSSL*** segítségével régi alkalmazásaink hogyan felelhetnek meg a korszerű követelményeknek, vagyis hogyan tudnak kellő biztonságot nyújtani. Hogy mi a helyzet a vezeték nélküli hálózatokkal? Aki kénytelen nyílt szövegekkel dolgozó hálózati alkalmazásokat gyengén védett, vezeték nélküli, például 802.11b hálózatokon keresztül használni, az nyugodtabban fog aludni, miután használatba vette az Stunnelt? Hamarosan ez is kiderül.

A jelszó nélküli tanúsítványok használatának veszélyei

A félig vagy teljesen önműködően, parancsfájl által létrehozott kiszolgáló tanúsítványok használata gyors és kényelmes megoldás, de van vele egy kis baj: az ilyen tanúsítványokhoz szinte soha nem tartozik jelszó. Az engedélyek módosításával gondoskodnunk kell arról is, hogy a tanúsítványt csak a root tudja olvasni, így legalább rendszerünk jogok nélküli használói nem tudnak hozzáférni. Azt is érdemes viszont figyelembe venni, hogyha rendszerünk root hozzáférését valaki megszerzi, akkor a jelszó nélküli tanúsítványt akár rossz célra is fel tudja használni.

Persze lehet, hogy ennek kockázata nem különösebben érdekel bennünket, és valóban, a jelszó nélküli kiszolgáló tanúsítványokat széles körben használják. Jobban belegondolva, elég bosszantó, ha az *Stunnel* összes indításakor kézzel kell beírni a jelszót. Természetesen a biztonsági szakemberek meglehetősen merész húzásnak tartják a jelszó nélküli tanúsítványok alkalmazását. Ha egy folyamat van annyira fontos, hogy a titkosítás kiemelt szerepet kapjon, akkor legyen annyira fontos, hogy minden alkalommal emberi közreműködéssel gondoskodunk az indításáról.

Háttérismeretek

Az ***Stunnel*** használatához két dologgal kell tisztában lenni. Először is, tudni kell, hogy a hálózati alkalmazások hogyan használják a hálózatot. Ha egy egyszerű, csupán egyetlen TCP-kaput használó alkalmazásról van szó, ilyen például a kizárólag a 23-as TCP-kapun át forgalmazó ***telnet***, akkor az ***Stunnel*** minden további nélkül működik. Ha UDP-t, kapumegfeleltetőt (***portmapper***) vagy egyéb dinamikus kapumegfeleltető módszert használ, akkor az ***Stunnellel*** nem jutunk előrébb. Az RPC alkalmazások például nem fognak működni, mert kapumegfeleltetőt használnak. Az FTP a 21-es TCP-kapun keresztül vezérli a forgalmat, de az adatkapcsolatokat magasabb sorszámú, véletlenszerűen kiválasztott kapukon át bonyolítja, így szintén kiesik a körből.

Másodszor, tisztában kell lenni a nyilvános kulcsú titkosítás alapjaival, bár az ***X.509***-es nyilvános kulcsú infrastruktúrát (***PKI***) nem feltétlenül kell ismerni. Ezekről több korábbi írásomban is volt már szó. (Például: „Az ***OpenSSH*** száz meg egy előnye”, Linuxvilág, 2001. február-márciusi szám). Most legyen elég annyi, hogy a nyilvános kulcsú titkosításnál

© Kiskapu Kft. Minden jog fenntartva

1. kódrészlet Kiszolgáló tanúsítvány létrehozása az OpenSSL használatával

```
helyiugyfel:/etc/stunnel# openssl req -x509
➥-newkey rsa:1024 -days 365
➥-keyout stunnel.pem -out stunnel.pem
Using configuration from
/usr/lib/ssl/openssl.cnf (A
/usr/lib/ssl/openssl.cnf fájlban megadott beál-
lítások használata)
Generating a 1024 bit RSA private key (1024 bi-
tes RSA titkos kulcs létrehozása)
...++++++
.........................................+++
+++
writing new private key to ´key2.pem´
(Az új kulcs kiírása a „key2.pem” fájlba)
Enter PEM pass phrase: (Adja meg a PEM jelszót:)
************
Verifying password - Enter PEM pass phrase:
(Jelszó ellenőrzése - Adja meg a PEM jelszót)
******
---
You are about to be asked to enter information
that will be incorporated into your certificate
request. (Az alábbiakban megadott adatok a tanú-
sítványkérelemben is szerepelni fognak.)
What you are about to enter is what is called
a Distinguished Name or a DN. (Következőként
a megkülönböztetett név, röviden DN megadására
lesz szükség.)
There are quite a few fields but you can leave
some blank. (Csak néhány értéket kell megadni,
és bizonyos mezők üresen is hagyhatók.)
For some fields there will be a default value,
(Egyes mezőknél alapértelmezett értéket lát
majd,)
If you enter ´.´, the field will be left blank.
(ilyenkor a „.” karaktert beírva hagyhatja üre-
sen a mezőt.)
---
Country Name (2 letter code) [AU]:US
(Országnév, kétbetűs kóddal)
State or Province Name (full name) []:Minnesota
(Állam, tartomány teljes neve)
Locality Name (eg, city) []:St. Paul
(Helység, pl. város neve)
Organization Name (eg, company) []:pelda
(Szervezet, pl. cég neve)
Organizational Unit Name (eg, section) []:
(Szervezeti egység, pl. osztály neve)
Common Name (eg, YOUR name)
[]:helyiugyfel.pelda.org
(Közös név, pl. az ön saját neve)
Email Address []:X.509kozpont@pelda.org

nearclient:/etc/stunnel# chmod 600 stunnel.pem
```

minden résztvevőnek két kulcsa van: egy nyilvános, amelyet megoszt a többi résztvevővel és egy titkos, amit megőriz magának. A többi fél a mi nyilvános kulcsunkat használja a nekünk szóló üzenetek rejtjelezésére, mi pedig titkos kulcsunkkal fejtjük azokat vissza.
A digitális aláírások pontosan fordítva működnek. Ha aláírunk valamit a titkos kulcsunkkal, akkor nyilvános kulcsunk birtokában bárki ellenőrizheti, hogy az aláírás létrehozása a mi titkos kulcsunk felhasználásával történt-e, vagyis lényegében mi voltunk-e az aláírók. Ismétlem, a rendszer működésének alapja, hogy a titkos kulcsot csak mi ismerjük, függetlenül attól, hogy nyilvános kulcsunkból hány másolat létezik a világon.
Az ***X.509*** világában a nyilvános kulcsot tanúsítványnak hívjuk, amely lényegében a titkos kulcsból és a tulajdonos személyes adataiból, például nevéből és e-mail címéből álló adathalmaz, digitális aláírással ellátva. A titkos kulcsot egyszerűen csak kulcsnak nevezzük. Hogy növeljük a keveredést, előfordul, hogy a kettőt együttesen ugyancsak tanúsítványnak nevezzük. A szövegkörnyezet szerencsére mindig segítségünkre van: ha jelszótól mentes tanúsítványt említünk, akkor tudjuk, hogy kombinált kulcsról/tanúsítványról van szó, mivel maga a tanúsítvány, mint a nyilvános kulcs, nem rendelkezhet jelszóval.
Azt hiszem, ennél rövidebben még sosem sikerült összefoglalnom a nyilvános kulcsú titkosítás és az ***X.509*** lényegét. Ha mindez nem elég a cikk további részeinek – stílusosan szólva – dekódolásához, akkor tanulmányozzuk át az ***Stunnel GYK***-t vagy a mindenre kiterjedő ***RSA Crypto GYK***-t. (Lásd az internetes források részt.) Nos, itt az ideje, hogy végre az ***Stunnellel*** is elkezdjünk foglalkozni.

Az Stunnel telepítése

Jó esélyünk van rá, hogy Linux-terjesztésünk tartalmaz bináris ***Stunnel*** csomagokat. Az újabb ***SuSE***, ***Fedora*** és ***Red Hat Enterprise*** terjesztésekben az ***Stunnel 4***-es, a ***Debian 3.0***-ban (***Woody***) pedig 3.22-es változata szerepel.
A 3.22-es biztos és áttekinthető működésű változat, kiváló leírások tartoznak hozzá, míg a négyesben számos részt újraírtak, amelynek köszönhetően több alagutat is könnyebben lehet kezelni vele. A továbbiakban az újabb változattal foglalkozok. Aki ***Debiant*** használ, erősen fontolja meg az újabb ***Stunnel*** forrásának letöltését és lefordítását.
Az ***Stunnelt*** bármelyik terjesztés alatt könnyen és gyorsan le lehet fordítani. Először ellenőrizzük, hogy telepítettük-e terjesztésünk ***OpenSSL*** csomagját (ennek neve általában ***openssl***), az ***OpenSSL*** fejlesztői könyvtárakat (***openssl-devel*** vagy ***libssl096-dev***) és a TCP-burkoló fejlesztői könyvtárakat (***Debianon libwrap0-dev***, a ***SuSE*** és ***Fedora*** alaptelepítéseknek pedig része). Ezután bontsuk ki az ***Stunnel*** forrás .tar állományát, és adjuk ki a megszokott parancsokat:

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

Ha valami nem működik megfelelően, próbálkozzunk a `./configure –help` paranccsal, amely megjeleníti a különleges, a beállító parancsfájlnak átadható fordítási kapcsolókat. Ha végeztünk az ***Stunnel*** telepítésével, létre kell hoznunk néhány tanúsítványt, majd meg is kezdhetjük az alagutak használatát.

2. kódrészlet Átfogó beállítások az stunnel.conf fájlban

```
cert = /etc/stunnel/stunnel.pem
chroot = /var/run/stunnel/
pid = /stunnel.pid
setuid = nobody
setgid = nogroup
debug = 7
output = /var/log/stunnel.log
client = yes
```

A továbbiak nagy része csak az ***Stunnel 4.0.0*** és újabb változataira érvényes. Aki úgy dönt, hogy inkább a ***Debianban*** lévő 3.22-es ***Stunnel*** csomagot használja, az tanulmányozza át a csomaghoz mellékelt leírásokat vagy az ***Stunnel*** webhelyen szereplő példákat (lásd a forrásokat). A webhely anyagai túlnyomó részt a régebbi kiadást tárgyalják.

Tanúsítványok létrehozása az OpenSSL-lel

Minden ***Stunnel*** kiszolgálónak – vagyis olyan állomásnak, amely fogadja a valamelyik helyi nyílt szöveges szolgáltatásnak szóló titkosított csomagokat – szüksége van egy kiszolgáló tanúsítványra. Az ***Stunnel*** ügyfelek oldalán ilyesmire nincs szükség, hacsak nem akarjuk ügyféltanúsítvány alapú hitelesítésre állítani az ***Stunnel*** kiszolgálót. Az ügyféltanúsítvány alapú hitelesítés sajnos túlmutat jelenlegi témánkon, ezért egyik példában sem fog az ügyfél saját tanúsítvánnyal rendelkezni, ilyen csak a kiszolgáló oldalán lesz.
Ha az ***Stunnelt*** forrásból való fordítás után telepítettük, és ennek során kiadtuk a `make install` parancsot, akkor már van is kiszolgáló tanúsítványunk a (***/usr/local/etc/stunnel/stunnel.pem***) helyen. Ha a telepítést bináris csomagból végeztük, akkor lehet, hogy a csomag telepítés utáni teendőket ellátó parancsfájlja létrehozta számunkra a kiszolgáló tanúsítványt – de lehet, hogy nem.
A ***Fedora Core 1 Stunnel RPM***-je például – ki tudja, miért – üres tanúsítványt állít elő. Nyilván nem kell mondani, hogy helyes tanúsítványra van szükségünk. ***Fedora*** alatt ilyet úgy tudunk előállítani, hogy belépünk a ***/usr/share/ssl/certs*** könyvtárba, kiadjuk a `make stunnel.pem` parancsot, majd az ***stunnel.pem*** fájlt átmásoljuk a ***/etc/stunnel*** könyvtárba. Ehhez a tanúsítványhoz nem fog jelszó tartozni, ahogy az ***Stunnel*** forrás csomagjának `Make` parancsfájlja által létrehozotthoz sem tartozik.
Ha ***Stunnel*** kiszolgálónkhoz valamiért szükségünk van egy tanúsítványra, esetleg az önműködően létrehozott nem felel meg az igényeinknek – például, mert nem tartozik hozzá jelszó –, akkor az ***OpenSSL*** paranccsal sajátot is elő tudunk állítani. Az egyébként nagy tudású program használatát részletesebben – helyhiány okán – nem ismertetem. Szerencsére az ***Stunnel*** GYK (lásd a forrásokat) az ***OpenSSL*** és az ***Stunnel*** párbeszédével kapcsolatosan leggyakrabban felmerülő kérdésekre megadja a válaszokat, illetve további, az ***OpenSSL***-lel kapcsolatos információforrásokra is tartalmaz hivatkozásokat. Csak annyit mondanék, hogy lehetnek olyan helyzetek, amikor érdemes saját hitelesítő szervezetet (CA) létrehozni, telepíteni a megfelelő CA tanúsítványt (nem a kulcsot) az összes ***Stunnelt*** futtató rendszerre, majd ezeket a CA tanúsítványokat használni az összes kiszolgáló tanúsítvány aláírására. Például akkor van erre szükség, ha azt akarjuk, hogy az ***Stunnel*** kiszolgálók csak a megadott tanúsítványok birtokában lévő ***Stunnel*** ügyfélgépektől fogadják a csatlakozási kéréseket. Sok, sőt a legtöbb ***Stunnel*** felhasználó számára a saját aláírással ellátott tanúsítványok is tökéletesen megfelelnek, ilyenkor nincs gond a CA üzemeltetésével. Nézzük tehát, hogyan lehet saját aláírású kiszolgáló tanúsítványt készíteni az ***OpenSSL***-lel.

3. kódrészlet Szolgáltatások megadása a helyiugyfelen

```
[telnets]
accept = 23
connect = tavolikiszolgalo.pelda.org:992
```

A lényeg az *1. kódrészlet* első sorában található. A paranccsal, ha a kapcsolóin balról jobbra haladunk végig, arra utasítjuk az ***OpenSSL***-t, hogy egy ***X.509*** digitális tanúsítvány formátumú tanúsítványkérelmet állítson elő, 1024 bites kulccsal dolgozó ***RSA*** titkosítási eljárást használjon, a tanúsítvány 365 napig legyen érvényes, a kulcsot és a – nyilvános – tanúsítványt ugyanabba a fájlba (***stunnel.pem***) írja ki. Ha inkább jelszómentes tanúsítványt szeretnénk létrehozni, akkor a sor végére illesszük oda a `-nodes` kapcsolót is. Előbb mindenképpen érdemes átolvasni az „A jelszómentes tanúsítványok használatának veszélyei” című széljegyzetet.
Az 1. kódrészlet fennmaradó része az ***OpenSSL***-lel folytatott párbeszédet szemlélteti. A program bekéri a tanúsítványba beillesztendő, az ***X.509*** szabvány által megadott egyéb adatokat, mint például az ország és a régió nevét. Ezt elrontani nem nagyon lehet, viszont a ***Common Name*** (közös név) megadása nagyon fontos. Ha kiszolgáló tanúsítványt hozunk létre, akkor a benne szereplő közös névnek annak az állomásnak a teljesen minősített tartománynevével, vagyis teljes állomásnevével kell egyeznie, amely a tanúsítványt használni fogja.
Az ***SSL/TLS***-képes alkalmazások nagy része a közös név alapján indít ***DNS***-kérést, és szerzi be a kiszolgáló IP-címét. Ugyan az ***Stunnel*** ezt nem teszi meg, amíg a beállítások módosításával rá nem vesszük, ám a közös név azonossá tétele a teljesen minősített tartománynévvel mindenképpen jó szokás.
Az 1. kódrészlet utolsó sorában `0600`-ra (`-rw-------`) módosítottam az új kiszolgáló tanúsítványra vonatkozó engedélyeket. Mivel az ***OpenSSL***-t rootként futtattam, tulajdonosa már eleve ő volt. Minden kiszolgálói tanúsítványnál fontos, legyen az akár jelszóval ellátott, vagy jelszómentes, hogy csak a root tudjon hozzáférni, és ő is csak olvasni tudja. Jómagam az ***OpenSSL*** parancsot a ***/etc/stunnel*** könyvtárból futtattam, a tanúsítványom tehát egyből a helyére került, és nem volt szükség arra, hogy kézzel helyezzem át.

Az Stunnel átfogó beállításainak megadása

Ha előállítottuk a megfelelő kiszolgáló tanúsítványt, állítsunk be magunknak egy alagutat. Ez a lépés tagadhatatlanul könnyebb lesz, mint az előző, viszont a programváltozatoktól is erősebben függ. Az ***Stunnel 4.0*** előtti változatai minden beállítást parancssori kapcsolóként fogadtak. A je-

© Kiskapu Kft. Minden jog fenntartva

© Kiskapu Kft. Minden jog fenntartva

4. kódrészlet Szolgáltatások megadása a tavolikiszolgalon

```
[telnets]
accept = 992
connect = 23
```

lenlegi változatban az egyetlen létező parancssori kapcsoló az, amely szükség esetén a beállító fájl alapértelmezettől eltérő helyét adja meg.

Ha az ***Stunnelt*** forrásból, az alapértelmezett fordítási beállításokkal telepítettük, akkor a ***/usr/local/etc/stunnel*** könyvtárban keresi a beállító fájlt. Ha a telepítést bináris csomagból végeztük, akkor nagy valószínűséggel a ***/etc/stunnel*** könyvtárban kell keresgélnünk. A 2. *kódrészlet* egy rövidített minta, mely az ***stunnel.conf*** fájl átfogó beállításait tartalmazza (a rövidítés leginkább a megjegyzések eltávolítására terjedt ki).

A `cert` átadott érték azt közli az ***Stunnellel***, hogy hol keresse kiszolgáló tanúsítványát; logikusan ilyesmire csak az ***Stunnel*** kiszolgálón van szükség, az ügyfeleken nincs.

A `chroot` érték azt a könyvtárat adja meg, amelybe indítás után az ***Stunnel*** `chroot` műveletet hajt végre – persze csak azután, hogy beolvasta a beállító fájlt és a kiszolgáló tanúsítvány fájlokat. Lehet, hogy a `chroot` helyszínéül szolgáló börtönt (***jail***) érdemes kézzel létrehozni, majd feltölteni néhány hasznos dologgal, például külön ***/etc/hosts.allow*** és ***/etc/hosts.deny*** fájlokkal, már amennyiben TCP-burkoló stílusú hozzáférés-vezérlést akarunk használni.

A `pid` azt határozza meg, hogy az ***Stunnel*** hova írja ki folyamatazonosítóját. Az itt megadott útvonal a `chroot` révén megadotthoz viszonyított, vagyis az ***Stunnel*** csak a `chroot` művelet végrehajtása után írja ki folyamatazonosítóját.

A `setuid` és a `setgid` adja meg, hogy az ***Stunnel*** melyik felhasználó és csoport nevében fusson. Ha az ***Stunnelnek*** 1025 alatti számú TCP-kapun is hallgatóznia kell, akkor rootként kell indítani. Ilyenkor beolvassa a beállító fájlt és a kiszolgáló tanúsítványt, köt a védett kapuhoz, majd lefokozza önmagát.

Alapesetben az ***Stunnel*** biztonsági értesítéseit és ennél nagyobb fontosságú naplóüzeneteit a helyi démon ***syslog*** eszköznek továbbítja. A ***Fedorában*** található változat üzeneteit az `authpriv` eszköznek küldi, innen ezek a ***/var/log/secure*** fájlba kerülnek. A `debug` kapcsolóval eltérő naplózási szintet is választhatunk. A hetes a legmagasabb szint, ezt akkor érdemes használni, ha nem sikerül munkára bírnunk az ***Stunnelt***. Az output beállítással arra vehetjük rá az ***Stunnelt***, hogy üzeneteit a megadott fájlba írja, és ne adja át a ***syslognak***.

A 2. kódrészlet utolsó sorában a `client` (ügyfél) kapcsolót „`yes`”-re, azaz igenre állítjuk, amivel jelezzük, hogy a megadott rendszer kezdeményezni fogja az ***SSL***-tranzakciókat, és nem fogadni. Értelemszerűen a kapcsolatokat fogadó kiszolgálón ez az érték az alapértelmezett „`no`”, vagyis „nem” marad.

Alagút megadása

Lassan célba érünk, végre az alagutakkal is foglalkozhatunk. Példaképpen ***telnet*** kapcsolat létesítése céljából hozzunk létre egy alagutat a `helyiugyfel` és a `tavolikiszolgalo` állomások között. A `tavolikiszolgalo` ***stunnel.conf*** fájljának

5. kódrészlet Példa telnet kapcsolat

```
helyiugyfel:/usr/local/etc/stunnel#
➥telnet 127.0.0.1
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is ´^]´.
Fedora Core release 1 (Yarrow)
Kernel 2.4.22-1.2115.nptl on an i686
login: tavoli_felhasznalonev
Password: **********
Last login: Sun Jun 13 21:39:17 from
localhost.localdomain
[tavoli_felhasznalonev@tavolikiszolgalo
➥tavoli_felhasznalonev]$
```

átfogó beállításai gyakorlatilag meg is egyezhetnek a 2. kódrészletben szereplőkkel, azzal a kivétellel, hogy a `client` beállításnak „`no`” értéket kell adni. A két állomás beállításai között a legfontosabb különbség a szolgáltatások megadásában lelhető fel.

Mielőtt mélyebbre ásnánk, ismerkedjünk meg részletesebben is a példakörnyezettel. Tegyük fel, hogy a `tavolikiszolgalo` már ***telnet*** kiszolgálóként üzemel, vagyis már képes a ***telnet*** kapcsolatok fogadására a 23-as TCP-kapun keresztül. Ha nem akarjuk, hogy a `helyiugyfel` a nyílt szövegű forgalmat bonyolító kapura csatlakozzon, akkor az ***SSL***-kapcsolatok számára másik kaput kell választanunk. Micsoda szerencse, hogy az ***IANA*** már kiválasztotta az ***SSL*** alapú ***telnet***, vagyis ***telnets*** kapcsolatok kapuját, ez a 992-es TCP-kapu.

Az alagútnak tehát a `helyiugyfel` és a `tavolikiszolgalo` 992-es TCP-kapuja között kell létrejönnie. De vajon honnan tudja az SSL kezelésére képtelen ***telnet*** parancsunk és hasonló hiányosságokkal küzdő ***telnet*** kiszolgálónk, hogy hogyan használja az alagutat? A kérdés csapda volt, az alagút ugyanis a küldő és a fogadó ***telnet*** folyamat számára egyaránt tökéletesen átlátszó.

A `helyiugyfel` ***Stunnel*** folyamata a szokásos kapun, a 23-as TCP-n fogadja a kapcsolatot (a kapu számát meg lehet változtatni), SSL alapon titkosítja a csomagokat, majd továbbítja őket a `tavolikiszolgalo` 992-es TCP-kapujára.

A `tavolikiszolgalo` visszafejti a csomagokat, majd továbbítja őket a 23-as TCP-kaput figyelő helyi ***telnet*** folyamatnak. Persze a csomagok előbb a ***xinetd***-hez vagy az ***inetd***-hez kerülnek, az ***in.telnetd*** csak utána kapja meg őket, de a lényeg – remélem – érthető.

Ennél a megoldásnál, ha a `helyiugyfel` felhasználói csatlakozni szeretnének a `tavolikiszolgalohoz`, akkor kiadják a `telnet 127.0.0.1` parancsot, aminek hatására létrejön a titkosított kapcsolat. A `tavolikiszolgalo` válaszcsomagjai ugyanezen az útvonalon haladnak, csak értelemszerűen ellentétes irányba. Mindkét ***telnet*** folyamat, a ***telnet*** ügyfél és az ***in.telnetd*** is azt hiszi, hogy helyi felhasználóval áll kapcsolatban, ám a csomagok a valóságban egy SSL titkosítású ***Stunnel*** kapcsolaton keresztül utaznak. Nagyjából ennyi a magyarázata annak az összesen hat sornak, amely a *3.* és a *4. kódrészletben* látható.

© Kiskapu Kft. Minden jog fenntartva

Mint kitűnik, a szolgáltatások megadásához mindössze két-két sorra van szükség, az `accept` sor a figyelt kaput adja meg, a `connect` sor pedig a célkaput. A két sor pontos jelentése attól függ, hogy ***Stunnel*** ügyfélről vagy ***Stunnel*** kiszolgálóról van-e szó. Az ügyfélrendszereken (`client = yes`) az ***Stunnel*** nyílt szöveges csomagokat vár a figyelt kapun, és titkosított csomagokat továbbít a célkapura. A kiszolgálókon (`client = no`) az ***Stunnel SSL***-kapcsolatokat, vagyis titkosított csomagokat fogad a figyelt kapun, és ezeket visszafejtett formában továbbítja a célkapura.

Érdemes megemlíteni, hogy az ügyfeleken a `connect` sorban nemcsak kaput, de egy távoli állomás nevét vagy IP-címét is meg lehet adni, ahogy a 3. kódrészletben is `connect = tavolikiszolgalo.pelda.org:992` szerepel. A 3. és 4. kódrészlet beállításai alapján tetszőleges állomás csatlakozhat a `helyiugyfel` 23-as TCP-kapujára, majd az alagúton keresztül kapcsolatba léphet a `tavolikiszolgaloval`. Ezt többféle módszerrel is meg tudjuk akadályozni, többek közt az ***iptables*** használatával. Ha a `helyiugyfelen` az `accept = 127.0.0.1:23` sort adjuk meg, akkor az ***Stunnel*** ügyfélfolyamatát rá tudjuk venni arra, hogy csak helyi kapcsolatokat fogadjon.

A megoldás a `tavolikiszolgalon` is működik. Ha a `tavolikiszolgalonak` több hálózati csatolója is van, például ***eth0***, ***wlan0***, ***ppp0*** stb., akkor hasonló `accept` utasítással választhatjuk ki azt a csatoló IP-címet, amelyen fogadni akarjuk az alagúton keresztül érkező csomagokat. Az ***Stunnel*** ügyfeleken és kiszolgálókon az `accept` utasítások IP-címe alapesetben az `any` (vagyis az összes helyi csatoló), a `connect` utasítás alapértelmezett IP-címe pedig a `127.0.0.1` (`localhost`).

Mit kell tudni a `tavolikiszolgalon` futó ***telnet*** kiszolgálóról? Végül is a nyílt szöveges ***telnet*** kapcsolatok használata általában elég rossz ötlet. Éppen ezért ne feledjük el a `bind = 127.0.0.1` sort hozzáadni a */etc/xinetd/telnet* parancsfájlhoz, ennek hatására csak helyi folyamatok kapcsolódhatnak a 23-as TCP-kapuhoz.

Ha az ügyfélen és a kiszolgálón egyaránt megtettük a szükséges beállításokat, akkor egyszerűen az `stunnel` parancs kiadásával indítsuk el az ***Stunnelt***. Különösebben nem kell foglalkoznunk azzal, hogy a kiszolgálón vagy az ügyfélen indítjuk el előbb, az ügyfél úgysem próbál alagutat létrehozni, amíg arra tényleges igény nincs, vagyis amíg – például – el nem indítunk egy ***telnet*** kapcsolatot. Ha valamelyik, esetleg mindkét fél jelszóval védett kiszolgáló tanúsítványt használ, akkor itt az ideje, hogy beírjuk a jelszót. Erről akkor se feledkezzünk el, ha az Stunnel indítására parancsfájlt készítünk. Ha a tanúsítvány beolvasása megtörtént, elvileg minden készen áll.

Adjuk ki a `ps auxw` parancsot, és ellenőrizzük, hogy a kimenetben szerepel-e az ***Stunnel*** folyamat. Maga az ***Stunnel*** a konzolra nem ír ki semmit, amivel visszaigazolná gond nélküli elindulásának tényét. Az erre utaló és egyéb jelzéseket, köztük az indítási üzeneteket a ***syslognak*** küldi el.

Ha az ***Stunnel*** az ügyfélen és a kiszolgálón is fut, akkor a `helyiugyfel` felhasználói a `helyiugyfel` 23-as TCP-kapujára – például a `telnet 127.0.0.1` paranccsal – csatlakozva válthatják ki az alagút létrehozását. Az *5. kódrészlet* egy titkosított ***telnet*** kapcsolat felépülését szemlélteti.

Az Stunnel és az inetd/xinetd kapcsolata

Gyónnom kell. Az itt szereplő példák nem mutatják be a *telnet* kapcsolatok *Stunnel* segítségével végzett titkosításának leghatékonyabb módját, bár azt legalább megígérhetem, hogy kipróbáltam őket, és működnek.

Azzal szeretném menteni magam, hogy szerettem volna egyszerűen ismertetni az általános TCP alapú szolgáltatások alagútba terelését, és olyan módszert akartam bemutatni, ami bármely egyetlen TCP-kaput használó szolgáltatással működik. A telnet, a pop3 és sok más szolgáltatás indítását viszont egy szuperkiszolgáló – *inetd* vagy *xinetd* – végzi, és az *Stunnel* számos különböző, itt nem tárgyalt szolgáltatással támogatja ezeket.

Megoldható például hogy az *Stunnel* kiszolgáló ne továbbítsa egy connect (csatlakozás) utasítással a csomagokat, hanem adja át őket egy másik folyamatnak, amelyet egy exec hívással az *stunnel.conf* alapján maga indít el. Az *Stunnel* maga is meghívható dinamikusan *inetd* vagy *xinetd* alól.

Az *Stunnel* dinamikusan indított démonokkal való együttes, illetve *inetd* vagy *xinetd* alatti használatáról az *stunnel(8) man* oldal és az *Stunnel GYK* tartalmaz további tájékoztatást.

Látható, hogy a felhasználó szempontjából a `tavolikiszolgalohoz` való kapcsolódás pontosan úgy zajlik, mint bármely más ***telnet*** kapcsolat létrehozása. A kódrészlet végén megjelenő ***bash*** parancssor jelzi, hogy a `tavolikiszolgaloval` valóban sikeresen létrejött az összeköttetés.

Összefoglalás

Remélem, az eddigiek alapján mindenki el tudja kezdeni az ***Stunnel*** használatát, illetve a vele való ismerkedést. Mint máskor, most is csak a felszínt érintettük. Mindenkinek meghagyom azt az örömöt, hogy önállóan fedezhesse fel az ***Stunnel*** ügyféltanúsítványok alapján történő alagúthitelesítési képességét, a TCP-burkoló stílusú hozzáférésvezérlés támogatását, valamint az ***stunnel.conf*** fájlba beilleszthető átfogó és a szolgáltatásokra egyedileg jellemző beállítások sokaságát.

Ennek során bátran forduljunk segítségért az ***stunnel(8) man*** oldalhoz – mire a végére érünk, megszokott, titkosításra képtelen, TCP alapú alkalmazásaink forgalmát többé nem lehet majd lehallgatni.

Linux Journal 2004. szeptember, 125. szám

Mick Bauer (mick@visi.com)
Biztonsági szakember, a Linux Journal biztonsági témákkal foglalkozó szerkesztője, biztonsági tanácsadó a Minnesota állambeli Minneapolisban található Upstream Solutions LLC Inc.-nél.

© Kiskapu Kft. Minden jog fenntartva

Az Arkeia 5.2 Network Backup

Központosított adatmentés Linux alapokon.

Az Arkeia-t 1999 áprilisa óta nem vizsgáltuk (*www.linuxjournal.com/article/3166*), ezért úgy gondoltuk, itt az ideje utánanézni, hogy mi újság ezzel a programmal.

A program szolgáltatásai

Az ***Arkeia Network Backup*** olyan ügyfél/kiszolgáló rendszerű adatmentő alkalmazás, amely akár vegyes hálózatok tagjainak biztonsági mentését is lehetővé teszi. A rendszer a másolatok tárolására Linux vagy UNIX alapú másolatkiszolgálót használ. A másolatkészítő rendszer ügyfélprogramjából számos különböző platformra készült változat létezik, köztük természetesen linuxos is. A különböző UNIX változatokon túl a program elérhető az olyan, bizonyos mértékig UNIX-szerű operációs rendszereken is mint amilyen a Mac OS X vagy a Microsoft Windows 98, ME, NT, 2003 és XP változatai.

Megfelelő bővítményekkel megoldható az olyan programok futás közbeni mentése (hot backup), mint az Oracle, Microsoft Exchange, Lotus Notes, IBM DB2 és MySQL.

Az ***Arkeia Disaster Recovery***, amely külön termék és most részletesen nem is vizsgáljuk, biztosítja a korábban mentett Linux ügyfelek és kiszolgálók „puszta vas” (bare-metal) állapotból való helyreállítását. Kipróbálásra mind a ***Network Backup***, mind a ***Disaster Recovery*** hozzáférhető 30 napos próbaverzió formájában. Egy harmadik termék, az ***Arkeia Lite***, amely egy linuxos kiszolgáló és mindössze két asztali rendszer biztonsági másolatának elkészítésére alkalmas, ingyenesen áll a felhasználók rendelkezésére.

Az Arkeia Network Backup 5.2.7 változatát vizsgáltuk, amelyet a ➲ ***www.arkeia.com*** címről töltöttünk le a PDF formátumú dokumentációval együtt. A linuxos változat a Debian GNU/Linux 2.2 és 3.0, a Mandrake 7.2–9.2, a Red Hat 6.0–9.0, a Slackware 8.0 és a SuSE 7.1–9.0 terjesztéseket támogatja.

Telepítés

A PDF formátumban letölthető dokumentáció mintegy ötszáz oldalnyi leírást tartalmaz, ami őszintén szólva elég ahhoz, hogy kicsit elbátortalanítsa az embert. A legrövidebb dokumentum a ***Quick Start Guide*** (***Gyorstalpaló az induláshoz***) címet viselte, ezzel kezdtem tehát az ismerkedést.

A kipróbáláshoz egy Debian 3.0-ás rendszert használtam. Az ***Arkeia Debian***-telepítője `.tar.gz` formátumban és nem Debian-csomagként érkezett. Ennek kicsomagolása után átváltottam a telepítőfa legfelső könyvtárára és kiadtam az `install` parancsot. Telepítés közben minden alapértelmezett beállítást elfogadtam.

Ezt követően elindítottam a `xarkeia` programot. Az *1. képen* is látható futurisztikus megjelenés eleinte kicsit szokatlanul hat. A leírás utasításait követve megadtam az ***Arkeia*** rendszergazdai jelszavát, elvégeztem a további szükséges beállításokat, és futtatni kezdtem egy látszólagos másolatkészítést. Amíg gondosan követtem az útmutatóban leírtakat, minden úgy történt, ahogy le volt írva. Egy ponton aztán kissé eltértem a folyamattól és anélkül próbáltam elindítani egy biztonsági mentést, hogy bármilyen szalagos eszközt beállítottam volna. A mentés megfeneklett és én sem beállítani nem tudtam a kívánt egységet, sem a grafikus felületen keresztül megszakítani a mentési folyamatot. Az akkori ismereteim kevésnek bizonyultak, hogy megoldjam a problémát. A támogatási weboldalon feltett kérdésemre fél órán belül választ kaptam, amely valóban tartalmazta a megoldást: egy démont kellett leállítani és újraindítani. A próbamentés ezek után minden további váratlan esemény nélkül lefutott, és befejeződött a telepítési folyamat. Az ***Arkeia*** készen állt a valódi biztonsági másolatok elkészítésére.

A felépítés

Az ***Arkeia*** a működéséhez szükséges adatokat egy adatbázisban tárolja , amelyben a rendszeradminisztrátor a következő dolgokat állítja be:

- Szalagos meghajtók (***tape drives***).
- Meghajtócsoportok (***drivepacks***): hasonló szalagos meghajtókból álló egységek.
- Szalagok (***tapes***): mindegyik saját címkével és előzményinformációkkal rendelkezik.
- Szalagkészletek (***tape pools***): hasonló szalagok csoportjai.
- Mentési csomagok (***savepacks***): az együtt mentett fájlok és könyvtárak csoportjai.
- Biztonsági másolatok (***backups***): egy adott szalagkészlet szalagjaira egy adott meghajtócsoport használatával felírt mentési csomag.
- Felhasználók (***users***): számos felhasználói szerep állítható be, ami lehetővé teszi, hogy egy nagyobb telephelyen a biztonsági másolatok elkészítésével kapcsolatos munkát több ember végezhesse.
- Kiszolgálók (***servers***): egy ***Arkeia***-rendszer több másolatkiszolgálót is tartalmazhat.

- Ügyfelek (*clients*): minden egyes kiszolgálóhoz több ügyfélrendszer csatlakozhat.

A biztonsági másolatok készítését és naplózását a másolatokért felelős kiszolgáló vezérli. A másolatkészítés egyaránt lehet kézzel vezérelt vagy önműködő – ez utóbbit az ***Arkeia Periodic*** (periodikus) kifejezéssel illeti. Megkülönböztetünk ezen kívül teljes és növekményes biztonsági másolatot. Utóbbinál azok a fájlok, amelyek egy megadott alapidőpont óta nem változtak, nem kerülnek archiválásra. A növekményes biztonsági másolatok naplózása több szintű rendszer szerint történik. Egy adott szint alapidőpontja a megelőző alacsonyabb szintű biztonsági másolat időpontja. A biztonsági másolat szintje egy adott periodikus archiválás minden fájlja szempontjából ugyanaz.
A szalagok kapacitásának optimális kihasználása és megtöltése végett ütemezhetjük úgy a másolatkészítést, hogy egy szalagra több mentés anyaga kerüljön, de kezdhetünk minden mentést új szalaggal is. A mentési csomagok fájlokat, adatbázisokat, könyvtárakat tartalmaznak. Egy ilyen csomagba több gépről származó anyagok is kerülhetnek. Az említett objektumok megfelelő bővítmények segítségével is archiválhatók. Ilyen például a MySQL-hez tartozó bővítmény.
A programkönyvtárak, tömörített állományok és a hasonló elemek számára különleges kezelőfelületek állnak rendelkezésünkre az ***Arkeia*** alatt, de úgy vannak beállítva, mint meghajtócsoportokba bejegyzett meghajtók. Így amikor egy archívumot az ***Arkeiával*** kezelünk, nem látunk nagyobb különbséget egy könyvtár és bármilyen más meghajtókészlet között.

A grafikus felület

Az ***Arkeiát*** vagy az `xarkeia` nevű grafikus ügyfélprogramon keresztül kezelhetjük, vagy parancssori ügyfélprogramok gyűjteménye segítségével. A tényleges tevékenységet az ezeken az ügyfélprogramokon keresztül elért démonok végzik. Valószínűleg sok rendszergazda gondolja úgy, hogy a grafikus felületen keresztül könnyebb elvégezni a beállításokat és a mentési vagy visszaállítási műveleteket.
A ***xarkeia*** grafikus felülete kizárólag az X rendszer szolgáltatásaira támaszkodik, vagyis nem használ sem ***Motif***-ot, sem ***Qt***-t, ***GTK***-t vagy bármilyen más, külső grafikus programkönyvtárat. A megjelenése éppen ezért olyan egyedülálló és minden mástól eltérő. Habár a környezethez való alkalmazkodás igényel egy kis időt, rövid gyakorlás után a használata nem okozhat gondot. Az ablakokat díszítő gombok, amelyeket a fejlécen szoktunk látni, most nincsenek ott. Ezeket a bal felső sarokban látható gombokból álló kör helyettesíti. Hiányoltam azt a lehetőséget, hogy a `xarkeia` egy példányát az egyik virtuális asztalról a másikra helyezzem át.
A `xarkeia` ablakának felső részén elhelyezkedő hibaüzenetpanel egy kis bosszúságot is okozott. Előfordultak ugyanis olyan hibaüzenetek, amelyek túl gyorsan törlődtek ahhoz, hogy alaposan átolvashattam volna őket. A gombokból álló kör `Help` (súgó) gombja környezetfüggő segítséget nyújt. Rendszertelen próbálgatással azt tapasztaltam, hogy az oldalaknak csak körülbelül a feléhez tartozott értelmes információ. Ezen a területen tehát még van hová fejlődni.
Ugyanakkor a tapasztaltabb rendszergazdák nyilván nem idegenkednek a kezelési utasításoktól, az ***Arkeia*** felhasználói

1. kép A Savepacks (mentési csomagok) menü beállítóképernyőjén láthatóak a haladó beállítások (advanced options) felett lévő sáv ikonjai, amelyekkel visszatérhetünk a menü gyökerébe

útmutatója pedig valóban teljes és alapos munkának tűnik.
A felhasználói felület testreszabásával kapcsolatban nem értem el túl sok eredményt. Amennyire meg tudtam állapítani, azokkal a betűtípusokkal és színekkel kell majd megbarátkoznunk, amelyeket a programmal együtt kapunk. Ezeken kívül semmilyen problémám nem volt a ***xarkeiával***. Számos kellemes tulajdonsága közül az egyik számomra legkedvesebb a ***Function Path Bar*** (műveleti útvonalsáv). Aki használt már olyan eszközt, amely számos menüszinttel rendelkezik, valószínűleg régen belefáradt már a keresgélésbe, és a folytonos visszalépésekbe. A `xarkeia` említett szolgáltatása – amint az 1. képen is látható – tárolja a menüfa alsóbb szintjeinek megközelítése közben használt ikonokat. A sáv valamelyik ikonjára kattintva így akár egyetlen mozdulattal több menüszintet is ugorhatunk visszafelé.

Egy valódi archiválás és visszaállítás

A ***Quick Start Guide*** elolvasása után a következő állomás az ***Arkeia User's Manual*** (Arkeia felhasználói kézikönyv) tanulmányozása. A maga 330 oldalával bizony bőven kínál olvasnivalót. Az `xpdf` segítségével megnyitottam a leírást és egy valódi mentés előkészítésével folytattam az ismerkedést. Ehhez egy ***Exabyte VXA*** meghajtót kívántam használni. Ennek az észlelése és beállítása nem okozott semmilyen problémát.
Meghatároztam egy új meghajtócsoportot, bejegyeztem és felcímkéztem egy csomó szalagot és felvettem azokat egy szalagkészletbe. A szalagcímkéző párbeszédablaknak egy „eject"-gombot kellett volna használnia. A ***Quick Start Guide*** gyakorlatainak köszönhetően már be volt állítva egy mentési csomagom, ezért interaktív mentést indítottam el, majd beállítottam egy periodikus mentést is többszöri ismétléssel, időt hagyva magamnak, hogy közben fájlokat adjak hozzá, vagy töröljek.
A ***Restoration*** (helyreállítás) menüben a fájlok kereséssel, vagy egy szokásos faszerkezetet megjelenítő böngészővel

© Kiskapu Kft. Minden jog fenntartva

© Kiskapu Kft. Minden jog fenntartva

jelölhetők ki. A ***Search*** (keresés) gomb megnyomása után előugró *„**Invalid regular expression!**”* (érvénytelen szabályos kifejezés) hibaüzenet egy kicsit megzavart, de csak addig, amíg el nem olvastam a felhasználói kézikönyv vonatkozó szakaszát, amely hangsúlyozza, hogy be kell jelölnöm néhány jelölőnégyzetet, valamint be kell írnom a keresési kifejezéseket a szomszédos szövegmezőbe. Mindezek alapján hasznosabb lett volna a „Legalább egy jelölőnégyzetet be kell jelölni” hibaüzenet.

A helyreállítás rengeteg beállítási lehetőséget tartalmaz. Megadhatjuk többek között a visszatöltés helyét, a tulajdonosi és hozzáférési jogokat, a létező fájlok felülírását, és a visszaállított fájlok ellenőrzésének módját. Az archívumböngésző segítségével kijelöltem a szükséges fájlokat és elindítottam a visszaállítást, de csak a *„**Please insert tape Monthly22**”* (Helyezze be a Monthly22 szalagot) üzenetig jutottam. Egy kis találgatás után szerencsére tényleg meg tudtam kezdeni a visszaállítást.

A szalagok indexelése

Az ***Arkeia*** hálózaton keresztül elérhető szalagindexet használ az előzmények és a beállítások tárolására, és erre támaszkodik helyreállítás esetén. Ez az index lehetővé teszi, hogy az adott dátumhoz tartozó archívumot hálózaton keresztül böngésszük, és a szükséges anyagokat egyszerűen visszaállítsuk. A módszer egyetlen hátulütője – és ez valamennyi ilyen megoldásra vonatkozik – hogy maga a hálózati index is elveszhet.

Az index abban a könyvtárban található, ahová az ***Arkeiát*** telepítettük. Ez alapértelmezésben a ***/opt/arkeia***, az index pedig a ***server/dbase*** alkönyvtárban található. Ha elveszik, a rendelkezésre álló segédprogramokkal a szalagokról ez is visszaállítható. Ehhez minden egyes szalagot be kell tölteni, ami nagyobb adatmennyiségnél meglehetősen hosszadalmas és fáradságos folyamat is lehet. Mindazonáltal az a tény, hogy az index helyreállítására egyáltalán van lehetőségünk, mindenképpen jó dolog. Használtam már korábban is hálózatos indexet alkalmazó archiváló programokat, de egyik sem volt képes erre. Adott esetben jól jöhet egy ilyen helyreállító módszer, habár a legjobb az, ha nem kerülünk ehhez hasonló helyzetbe.

Az ***Arkeia Disaster Recovery*** megkönnyíti az ilyen helyzetek kezelését is, hiszen készségesen elvégzi az archivumkiszolgáló vagy bármelyik ügyfélgép közvetlenül a szalagról történő „bare-metal” helyreállítását. Azoknak a megfontolt, de vállalkozó szellemű rendszergazdáknak, akik egy szabványos telepítés után a finomhangoláshoz szükséges adatfájlokat szalagról akarják betölteni a gyártó azt javasolja, hogy mindig legyen egy másolatuk az ***Arkeai*** telepítési könyvtáráról, valamint a ***server/dbase*** könyvtár tartalmáról. Ezek segítségével a helyreállítás még akkor is elvégezhető, ha maga a mentési kiszolgáló sérült meg. Minden esetben végezzünk vizsgálatot visszaállítás után, mivel a tapasztalatok helyről helyre változhatnak.

Egyebek

A program lehetővé teszi a szalagkettőzést is, ami akkor hasznos, ha a mentés egy példányát helyben, egy másikat pedig házon kívül, biztonságos helyen akarunk tárolni.

A parancssoros ügyfélprogramok, amelyeket részben a felhasználói útmutató befejező fejezetei, részben pedig a 137 oldalas ***Command Line Interface Manual*** (a parancssoros felület kezelési útmutatója) ismertetnek, lehetővé teszik az ***Arkeia*** parancssoron keresztül történő kezelését és módot adnak a program saját parancsfájlokkal történő elérésére is. Ehhez néhány mintát is ad az útmutató.

Áttekintés

Összességében megkedveltem a programot. A grafikus felület egy kicsit szokatlan, az illesztések és a kidolgozás néhol egy kicsit csiszolatlannak tűnik. A felépítés arról tanúskodik, hogy a tervezők alaposan végiggondolták, mire van szüksége egy biztonsági mentéseket végző programnak. A különböző platformok közötti átjárhatóság jónak mondható. A parancssori ügyfélprogramok kizárják annak lehetőségét, hogy egy lefagyott grafikus felület, amely nem enged betekintést a felszín alá, megakadályozza a rendszer mentését. A ***Quick Start Guide*** segítségével fáradtság nélkül sajátíthatjuk el a program kezelésének alapjait, a másik két kezelési útmutató pedig láthatólag erre épít, és teljessé teszi a képet. A 30 napos ingyenes próbaváltozat támogatása is gyors és hatékony volt.

Linux Journal 2004. július, 123. szám

Dan Wilder a Specialized Systems Consultants, Inc. műszaki igazgatója.

A termék adatlapja

Fejlesztő:	Arkeia
Honlap:	www.arkeia.com
Ár:	590-1190 dollár háromtól hét gépes rendszerekig, a nagyobb kiépítések ára 20.000 dollárig terjedhet a felépítéstől függően.

Előnyös tulajdonságok

- Többféle operációs rendszert támogat.
- Központilag ütemezett biztonsági mentések.
- Futás közben végrehajtott mentéseket támogató bővítmények.
- Böngészhető tartalomjegyzék a szalagokhoz.
- Jó felhasználói dokumentáció.

Hátrányok

- Nem szabványos grafikus felület.
- Eltűnő hibaüzenetek.
- Befejezetlen a környezetérzékeny súgó.

© Kiskapu Kft. Minden jog fenntartva

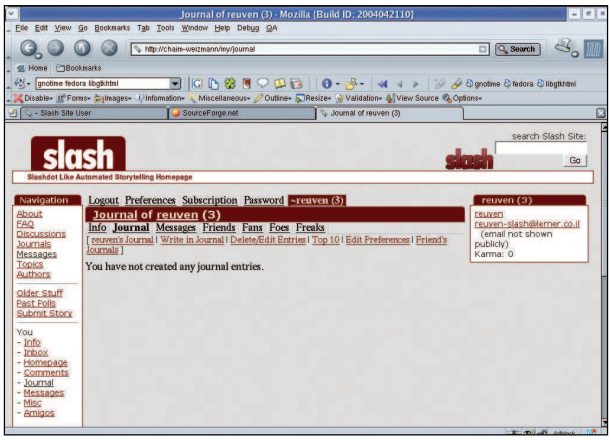
Weblogok és Slash

A Slash bonyolult, de harcedzett közösségi rendszerének funkciói közt megtaláljuk a megjegyzésekkel kiegészített felhasználói naplót is. Tegyük lehetővé felhasználóinknak, hogy barátok és ellenségek hálózatát hozzák létre és moderálják egymás megjegyzéseit.

Az augusztusi számban a Slashdot lapot is működtető Slash nyílt forráskódú Weblog rendszer telepítésével és karbantartásának alapjaival foglalkoztunk. A Slash a Perl, mod_perl és Apache előnyeit használja ki.

Napló készítése

A Slash a napló (journal) nevet használja a Weblog kifejezés helyett. A rendszer valamennyi felhasználója saját naplóval rendelkezhet; ezt a funkciót a „You” menü Journal pontja alatt érhetjük el, amelyet általában a képernyő bal oldalán találunk. A pont a journal.pl programot hívja meg, amelyet a lapunk Slash könyvtárában találunk. Az én chaim-weizmann nevű gépemen a `journal.pl` fájl a `/usr/local/slash/site/chaim-weizmann/htdocs/journal.pl` könyvtárban található. A kódot sokkal könnyebb olvasni mint gondoltam, de még a tapasztalt Perl guruk is úgy fogják találni, hogy a a Slash környezetben igen sok függvény központosított és testreszabott. Azt mondják a Slash rendszerét megváltoztatni nem különösebben bonyolult ha valaki hajlandó foglalkozni vele.
Amikor először kattintunk a Journal hivatkozásra, az *1. ábrán* látható képernyőképhez hasonló oldallal találkozunk. Az itt olvasható üzenet azt jelenti, hogy még nem készítettünk egyetlen naplóbejegyzést sem. Számos hivatkozás áll rendelkezésünkre, amennyiben a naplónkba kívánunk írni illetve módosítani szeretnénk a már meglévő bejegyzéseinket.
Készítsünk egy új naplóbejegyzést a „Write in Journal” (Írás a Naplóba) hivatkozásra kattintva. A 2. ábrán látható lapra jutunk. Írjuk be a címet, a témát (témák globális listájának gyűjteményét, a felhasználói naplóval együtt), valamint jelöljük meg, szeretnénk-e, hogy megjegyzéseket fűzhessenek a naplónkhoz végül gépeljük be magát a bejegyzést. Miután befejeztük a bejegyzés szerkesztését, rábökhetünk a ***Preview*** (előnézet) gombra, így elküldés előtt megtekinthetjük írásunkat. Számomra ez kicsit túlbiztosításnak tűnik; természetesen megértem, hogy az embereknek általában szeretik munkájuk hibátlanságát ellenőrizni elküldés előtt, de néha szeretnék gyorsabban haladni az előnézet megtekintése nélkül.



1. ábra Új Slash napló bejegyzések nélkül

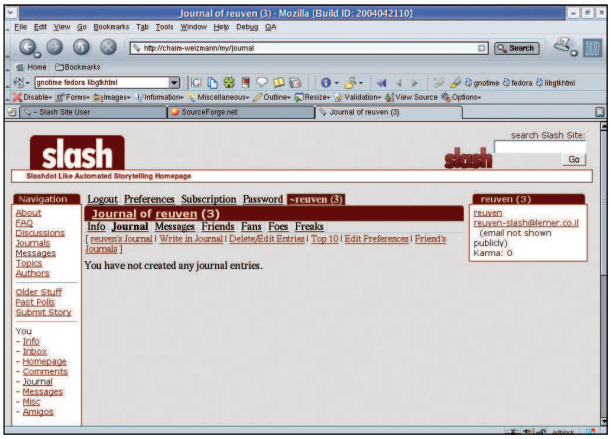
A ***Preview*** gomb után egy listát találunk, amellyel meghatározhatjuk, a naplóbejegyzésünk formázási tulajdonságait. Alapértelmezés szerint ez HTML formátum, lehetővé teszi, hogy HTML kódokat szúrjunk a naplónkba, például `<b>boldface</b>` és `<i>italic</i>` szövegeket. Mivel a HTML nem tesz különbséget az üres (whitespace) karakterek között, ennél a módszernél természetesen bekezdéseinket `<p>` tegekkel kell elválasztanunk egymástól. Egyúttal azt is jelenti, hogy a < vagy > karaktereket csak a megfelelő HTML jelölés (`&lt;` és `&rt;`) segítségével vihetjük be.
Magam részéről az extrans formázási módot választottam volna, feltéve, hogy már az elején tudom mit is jelent ez tulajdonképpen. Az extrans ugyanis feltételezi, hogy minden karaktert betű szerint kel érteni, majd a többszörös újsor karaktereket HTML bekezdéshatárrá alakítja. Észrevettem, hogy ez a lehetőség a HTML kódokat is szöveggé alakítja, de ez sokkal kevésbé tűnt érdekesnek, mint, hogy a végső másolat megőrzi a bekezdéshatárokat.
Miután legalább egyszer megtekintettük a bejegyzésünket, a ***Preview*** és a formázási választéklista közt megjelenik a ***Save*** (mentés) gomb. Folytathatjuk bejegyzésünk módosítását, megtekintését vagy az Add gomb segítségével

© Kiskapu Kft. Minden jog fenntartva

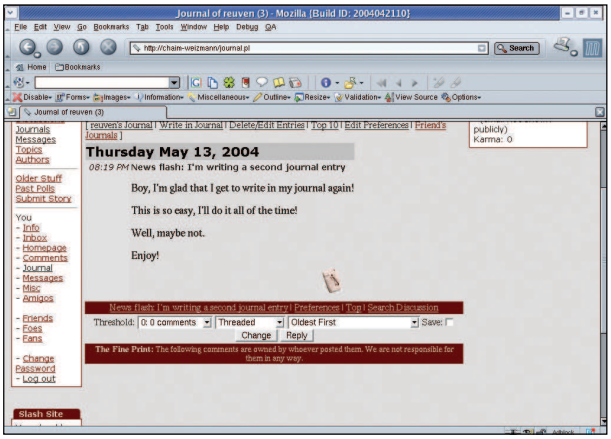
elmenthetjük, hogy mindenki láthassa azt. A `chaim-weizmann` gépen általam készített naplót például mindenki láthatja, aki a böngészőjét a `chaim-weizmann/~reuven/journal` URL-re irányítja.

Megjegyzések írása

Amint azt már más Weblog és naplóprogramoknál megszokhattuk, a Slash is lehetővé teszi, hogy más felhasználók megjegyzéseket fűzzenek írásunkhoz. Alapértelmezés szerint ez a lehetőség kikapcsolt állapotban van, az utasítások pedig egyértelműen és több ízben is felhívják rá a figyelmet, hogy ha egyszer bekapcsoljuk, akkor az örökre úgy is marad.
A lehetőséget minden egyes naplóbejegyzéshez egyenként kapcsolhatjuk be tetszés szerint. Az alapértelmezett beállítást az ***Edit Preferences*** hivatkozásra kattintva tudjuk megváltoztatni. Minthogy ez a beállítás csak az alapértelmezett értéket változtatja, a már létező naplóbejegyzéseinkre és megjegyzéseinkre természetesen nem vonatkozik.



2. ábra Napló beviteli oldal, ahol a naplónkba jegyzendő szöveget írhatjuk be



3. ábra Minden naplóbejegyzést a Slashdot-stílusú menü követ

A rendszerrel csak most ismerkedőknek nem teljesen magától értetődő, hogyan kell a megjegyzéseket engedélyező naplókhoz megjegyzéseket fűzni. Minden bejegyzést egy menüsor követ (3. ábra), amely vezérli a vita megjelenítését valamint lehetővé teszi a csatlakozást. Szerintem ez kicsit zavaró, hiszen nem nehéz összekeverni a ***Reply*** (válasz) gombot melynek segítségével hozzászólhatnánk a vitához és a menüsor többi részét, amely a vita megjelenítését változtatja.
Válasz küldése azért kicsit bonyolultabb ennél. Ha az eredeti küldeményre szeretnénk válaszolni, akkor a cikket közvetlenül követő ***Reply*** gombra kell kattintanunk. Amennyiben azonban egy megjegyzésre szeretnénk reagálni és ezzel új vitaszálat létrehozni, a ***Reply to This*** hivatkozásra kell kattintanunk, amelyet közvetlenül a válaszok alatt találunk meg. Ennek a szerkezetnek logikailag persze van értelme, de meg kell vallanom, hogy bár évekig követtem és használtam Slash-meghajtású vitákat, jó időbe telt, mire sikerült megértenem a két módszer közti különbséget.
A felülettől eltekintve a megjegyzés hozzáadása azonos az új küldemény felvételével, azzal a különbséggel, hogy nem tilthatjuk meg az embereknek az írásunkra adott választ. Gépeljünk be valamilyen címet majd a megjegyzést, jelöljük be a formázást, majd tekintsük meg az előnézetet illetve adjuk a vitához a lapot. A Slash lehetővé teszi, hogy a ***Anonymous Coward*** (Gyáva Anonymous) néven ismert anonymous felhasználóként adjunk fel küldeményt, ha a bejegyzés melletti „post anonymously" dobozra kattintunk. Sok rendszergazda azonban megtiltja rendszerén az ilyen anonymous küldeményeket, feltételezve, hogy a névtelenség csökkenti a felelősségérzetet.
Végül, a megjelenítési beállítások segítségével az egész vitát néhány mód bármelyikével megtekinthetjük. A threshold (küszöb) beállítás segítségével a megjegyzéseket szűrhetjük a lap közösségének más tagjai által adott pontszámok alapján. A moderálási és a hozzá kapcsolódó meta-moderálási képességeket a lap gazdája állíthatja be, így lehetővé téve a közösség tagjainak, hogy eldöntsék melyik megjegyzés érdemli a legtöbb figyelmet.

A megjelenítési beállítások a szálak megjelenését befolyásolják. A magam részéről egymásba ágyazott módon szeretem nézni az ilyen vitákat, ahol a válaszok állandóan láthatóak, csak éppen a szülőjükhöz képest kicsit beljebb kezdve. Alapértelmezés szerint a Slash lapok szálankénti módban jelenítik meg a bejegyzéseket, ami azt jelenti, hogy az adott megjegyzést kifejezetten kérni kell ha meg szeretnénk nézni. Végül a megjegyzéseket különböző sorrendben jeleníthetjük meg. A naplóbejegyzések mindig megjelennek, mégpedig Weblog-stílusban a legfrissebb bejegyzéssel kezdve és a legidősebbel fejezve be a listát. A cikkekre adott válaszok ellenben általában időrendben jelennek meg, azaz a legöregebb megjegyzést találjuk legfölül. Ezért aztán egy idő után le kénytelenek leszünk a lap aljára pörgetni, ha nyomon szeretnénk követni a vitát.

Naplóközösségek

Annak alapján amit eddig láttunk, a Slash egyszerű lehetőséget kínál több felhasználónak, hogy saját naplóját létrehozza és kezelje. Ugyanakkor a felhasználók és naplóik közt nincs igazi kölcsönhatás; mindenki el van szigetelve mindenki mástól.

© Kiskapu Kft. Minden jog fenntartva

A Slash rendszerét azonban a hálózati közösségek segítésére írták, így számos képességgel rendelkezik amelyek az együttműködést és a közös munkát segítik elő. Először is, a Slash a rendszer valamennyi naplójáról statisztikákat készít. A `journal.pl` (azaz a fő napló) lapon a Top 10 hivatkozásra kattintva a megtudhatjuk, melyik lapokat módosították a legutóbb, ki írta a legtöbbet és melyik barátunk írta a legtöbb bejegyzést.

A „barát” szó alatt nem közösségi tagot kell érteni. A Slash rendszerében minden felhasználó az összes többi felhasználót barátként és ellenségként osztályozhatja érdekes kapcsolatrendszer-hálózatot hozva létre ezzel az emberek közt, amely kicsit hasonlít, de azért nem azonos az Orkut és LinkedIn rendszerek hálózataihoz.

Amennyiben valakit barátként vagy ellenségként szeretnénk kijelölni, a legegyszerűbb módszer, ha belépünk a honlapjukra, amely általában `~felhasználónév` alakú. Az én rendszeremen tehát az `chaim-weizmann/~reuven` URL alatt bárki elérheti a saját honlapomat. Az adott személy felhasználói neve mellett található ikon mutatja, hogy az jelenleg éppen barát (mosolygó fej), ellenség (vörös szomorú fej) vagy semleges (ez az alapértelmezett, napszemüvegben és furcsa önelégült mosollyal jelenik meg). Az ikonra kattintva, megváltoztathatjuk a kapcsolatunkat.

Az egyik nagy különbség a Slash és más személyi hálózat és közösségi weblap rendszerek közt, hogy az ilyen kapcsolatok nyilvánosak. A Slash lapon bárki megnézheti, kik a barátaim és kik az ellenségeim. Bár ez minden bizonnyal feszélyez néhány embert, akik tartva a nyilvános kényelmetlenségektől, nem szívesen jelölnek meg ellenségeket, azért szó sincs róla, hogy a Slash ne tudna izgalmas személyes hálózatot és kapcsolatrendszert felépíteni.

Nem csak a barátokat nézhetjük meg, hanem barátok barátait is.

Minden ilyen kapcsolat egyoldalú; A lehet B barátja, de ettől még B lehet A ellensége. Amikor valakinek a honlapjára belépünk, nem csak az ellenségeit és barátait látjuk, hanem a rajongóit (akik barátnak jelölték be) és ellenzőit (akik ellenségnek jelölték őt).

A barátok listájának legnagyobb praktikus előnye, hogy barátaink naplóját a Slash nyomon követi és frissíti nekünk. A lapunk tetején található „Friend's Journals” hivatkozásra kattintva barátaink és naplóik jelennek meg. Ez tulajdonképpen könyvjelző vagy az RSS hírösszesítő megfelelője a Slash rendszerében. Azzal, hogy embereket a barátaink közé helyezünk, könnyedén nyomon követhetjük naplóik változásait.

Érdemes használni a Slash-t?

Az évek során többször is megnéztem a Slash-t és de egyszer sem hagyott bennem mély nyomokat. A kódot elég nehéz megérteni, a felhasználói felület csúnya és a képességek korlátozottak. A Slash-alapú lapok viszonylag rondák, bár ez a Template Toolkit használatának hála mostanra már megváltozott. A képességek továbbra is elég korlátozottak olyan más közösségi keretrendszerekhez és eszközkészletekhez mérve mint az Xoops vagy az OpenACS.

Ugyanakkor a Slash-t nem is széles igényekre tervezték; inkább korlátozott képességkészletet próbál megvalósítani, de azt jól. Ebben a tekintetben igen sikeresnek mondható. Az ➲ use.perl.org kiváló példa erre, ahol új cikkek mellett a felhasználók saját naplókat is tarthatnak.

Amennyiben korlátozott mértékű hírújságot és bejelentésrendszert szeretnénk, miközben sok felhasználónak akarjuk lehetővé tenni naplók készítését és követését, a Slash jó választás lehet.

Továbbá, el kell ismernem, hogy a kód sokat fejlődött az évek során; ma már meg lehet érteni mi is történik, és ha valaki tapasztalt Web/adatbázis guru akár módosíthatja vagy bővítheti a funkciókat. A Slash sok kényelmes függvénnyel rendelkezik, amelyek megismeréséhez kell egy kis idő, mielőtt nekiugranánk a módosításoknak. Mindazonáltal ez minden Web/adatbázis eszközre igaz, így nem lenne igazságos azt állítani, hogy a Slash bármivel is rosszabb ezen a téren.

Legfőbb kifogásom a Slash-el szemben, eltekintve a (CVS-ben maradó) terjesztési verzióval kapcsolatos problémáktól és a dokumentációtól, annyi, hogy az új képességek felvételéhez nincsen a Xoops, OpenACS és a Zope modulok és csomagok rendszeréhez hasonló szabványos felülete.

Összefoglalás

A Slash, sok más nyílt-forrású programhoz hasonlóan igen hatékony, jól méretezhető, újoncok számára nehezen telepíthető és gyengén dokumentált. Más csomagoktól eltérően, inkább a mélységre mint a szélességre koncentrál, más rendszerekhez képest több képességet kínálva, de feláldozva a bővíthetőséget és általánosíthatóságot. Ha azonban lapunk akár csak megközelíti a Slashdot által vonzott felhasználók és látogatók számát, bölcs dolog fontolóra venni használatát.

Linux Journal 2004. augusztus, 124. szám

Reuven M. Lerner, aki hosszú ideje dolgozik Web/database programozási tanácsadóként és fejlesztőként, jelenleg a Northwestern University egyetemének Learning Sciences karán első éves hallgató. Weblogját az altneuland.lerner.co.il, őt magát a reuven@lerner.co.il címen érhetjük el.

© Kiskapu Kft. Minden jog fenntartva

Tíz fontos parancs, amit minden linuxos fejlesztőnek ismernie kell

Ha más kódját kell karbantartanunk, néhány egyszerű segédprogrammal komolyan megkönnyíthetjük az életünket.

Ebben a cikkben néhány gyakorlatilag minden Linux-telepítésben megtalálható parancsot szeretnék ismertetni. Segítségükkel javíthatjuk kódjaink minőségét, divatos kifejezéssel élve termelékenyebbekké válhatunk. A lista összeállítását saját programozói tapasztalataim alapján végeztem, és olyan eszközökből áll, amelyeket magam is rendszeresen használok. Némelyik magának a kódnak az elkészítésében nyújt segítséget, mások hibakeresésre használhatók, megint mások pedig az ölünkbe pottyant idegen kód visszafejtését teszik lehetővé.

1. ctags

Aki szereti az integrált fejlesztői környezeteket (IDE), talán sosem hallott erről az eszközről, vagy ha hallott is, idejétmúltnak vélheti. Pedig egy a címkéket ismerő szerkesztő mindig is értékes eszköz marad.
Ha címkékkel látjuk el kódunkat, akkor például `vi` vagy ***Emacs*** alatt hiperszövegként tudjuk kezelni. (*1. ábra*) A kód minden objektuma egy a saját meghatározásához vezető hiperhivatkozást kap. Ha például `vi` alatt turkálunk a kódban, és szeretnénk tudni, hogy a „`pelda`” változó hol van megadva, beírjuk a `:ta foo` parancsot. Ha a kurzor éppen a változón pihen, elég a Ctrl+] kombinációt használnunk. A `vi`-t kevésbé kedvelők számára jó hír, hogy a `ctags` már nem csupán C kóddal és a `vi` szerkesztővel használható. A `ctags` GNU változata olyan címkéket állít elő, amelyek ***Emacs***, illetve sok egyéb, a címkefájlok felismerésére képes szerkesztő alatt is kezelhetők. A `ctags` a ***C***-n és a ***C++***-on kívül számos más nyelvet is ismer, olyat mint a ***Perl*** és a ***Python***, sőt, még vastervező nyelvekkel is boldogul, mint például a ***Verilog***. Segítségével emberi szem számára is könnyen átlátható utalásokat lehet készíteni, amelyek alapján könnyebb a kód megértése, áttekintése. Lehet, hogy a `ctags` használata szerkesztés közben még nem nagyon érdekel valakit, az utalásokat azonban mégis érdemes átnéznie. Ilyenkor a `ctags -x *.c*` parancsot kell kiadnia.
Én azért is szeretem ezt a kis programot, mert mindegy, hogy egy vagy száz fájlt vizsgáltatunk meg vele, mindenképpen hasznos adatokhoz jutunk, ellentétben sok IDE-vel, amelyek csak akkor érnek valamit, ha a teljes alkalmazást látják. Mivel a programok ellenőrzésére nem alkalmas, ha hibás bemenetet adunk neki, értelemszerűen a kimenete is hibás lesz.

2. strace

Az `strace` segítségével hibakereső és forráskód hiányában is kifürkészhetjük, pontosan mi történik egy program futása közben. Nekem például az abszolút kedvencem az olyan program, ami el sem indul, de nem mondja meg, hogy miért. Lehet, hogy valamilyen fájlt hiányol, de az is előfordulhat, hogy csak a jogosultságokkal van baja. Az `strace` elárulja nekünk, hogy mit csinál a program a háttérben, vagyis milyen rendszerhívásokat indít és ezek milyen eredménnyel járnak, egészen addig a pontig, míg a futása véget nem ér. A `fork` hívások követésére is képes.
Én úgy tapasztaltam, hogy az `strace` sokszor gyorsabban megadja az engem érdeklő válaszokat, mint bármelyik hibakereső; főleg, ha ismeretlen kóddal kell dolgoznom. Néha az is előfordul, hogy éles, hibakereső nélküli rendszeren kell valamilyen kód hibáit megkeresnem. Ilyenkor az `strace`-t előkapva elkerülhetem a rendszer foltozását vagy a kód `printf` hívásokkal való teleszórását. Íme egy egyszerű példa. Mi történik, ha egy normál felhasználó megpróbál törölni egy védett fájlt:

```
strace -o strace.out rm -f /etc/yp.conf
A kimenetből megtudjuk, mi volt a baj:
lstat64("/etc/yp.conf", {st_mode=S_IFREG|0644,
➥st_size=361, ...}) = 0
access("/etc/yp.conf", W_OK) = -1 EACCES
(Hozzáférés megtagadva)
unlink("/etc/yp.conf") = -1 EACCES
➥(Hozzáférés megtagadva)
```

Ha menet közben akarunk hibákat felderíteni a `strace` segítségével, futó folyamatokhoz is csatlakozhatunk vele. Tegyük fel például, egy folyamat egy csomó időt tölt valamivel. Az `strace -c -p folyamatazonosító` paranccsal pillanatok alatt megtudhatjuk, mi folyik odabent. Néhány másodperc után nyomjunk Ctrl-C-t, és az alábbihoz hasonló kiírást fogunk látni:

```
% time  seconds  usecs/call calls errors syscall
------ -------- ---------- ----- ------ --------------
 91.31  0.480456       3457   139         poll
  6.66  0.035025        361    97         write
  0.91  0.004794         16   304         futex
  0.52  0.002741         14   203         read
  0.31  0.001652          3   533         gettimeofday
  0.26  0.001361          4   374         ioctl
  0.01  0.000075          8    10         brk
  0.01  0.000064         64     1         clone
  0.00  0.000026         26     1         stat64
  0.00  0.000007          7     1         uname
  0.00  0.000005          5     1         sched_get_priority_max
  0.00  0.000002          2     1         sched_get_priority_min
------ -------- ---------- ----- ------ --------------
100.00  0.526208            1665         total
```

Ebben az esetben a várakozást a `poll` rendszerhívás idézte elő, valószínűleg itt egy foglalatra várt a folyamat.

3. fuser

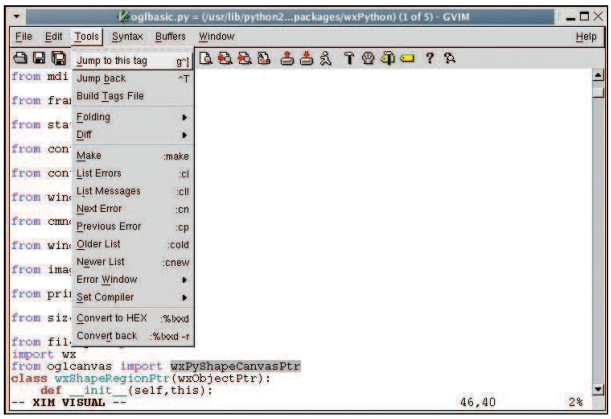
A név a „file user” (fájl használója) kifejezésből állt elő, vagyis a programmal azt tudhatjuk meg, hogy adott fájlt mely folyamatok nyitottak meg. Arra is alkalmas, hogy az összes ilyen folyamatnak jelzést küldjünk. Tegyük fel például, hogy törölni szeretnénk egy fájlt, de nem tudjuk megtenni, mert valamelyik program megnyitotta, és esze ágában sincs lezárni. Ilyenkor megúszhatjuk a gép újraindítását, ha kiadjuk a `fuser -k` fájl parancsot, ez `SIGTERM` jelzést küld az összes olyan folyamatnak, amely megnyitotta a fájlt.
Előfordulhat, hogy a `kill` paranccsal kell megölnünk egy `fork` hívásokkal – bármilyen okból – jónéhány példányra osztódott folyamatot. Egy kezdő programozó ilyenkor valószínűleg egy a célnak megfelelő `ps | grep` parancsot adna ki, majd szorgalmas másol-beilleszt játékba kezdene az egérrel. Sokkal egyszerűbb azonban, ha kiadjuk a `fuser -k ./program` parancsot, ahol a program a futtatható fájl elérési útja. A `fuser` általában a felügyeleti eszközök számára fenntartott `/sbin` könyvtárban található. Ha gondoljuk, a `/usr/sbin` és a `/sbin` könyvtárakat adjuk hozzá a `$PATH` környezeti változóhoz.

5. ps

A `ps`-t általában a folyamatok állapotának megvizsgálására használják, és sokan nem is tudják, hogy hibakereső eszközként is kiválóan megállja a helyét. Mindezeket a lehetőségeket a `-o` kapcsolóval érhetjük el, amely számos adatot bocsát rendelkezésünkre a folyamatokról, ideértve a processzorhasználatot, a képzetes memória iránti igényt, az aktuális állapotot stb. Mindezen mutatók jelentős része a ***POSIX*** szabványban is szerepel, vagyis többféle rendszeren is elérhető.
Ha a futó programokat, folyamatazonosítót és folyamatállapotot szeretnénk kilistázni, adjuk ki a
`ps -e -o pid,state,cmd`

parancsot. A kimenet így fog kinézni:

```
4576 S opt/OpenOffice.org1.1.0/program/
➥soffice.bin -writer
```



1. kép Címkékkel ellátott fájl szerkesztése gvimmel

```
4618 D dd if /dev/cdrom of /dev/null
4619 S bash
4645 R ps -e -o pid,state,cmd
```

A kimenetben láthatjuk, hogy a `dd` parancs nálam megszakíthatatlan alvó (`D`) állapotban volt. Lényegében arról van szó, hogy amíg a program vár a */dev/cdrom*-ra, addig blokkolt állapotba kerül. Az ***OpenOffice.org writer*** éppen alvó (`S`) állapotban van, a `ps` pedig futóban (`R`).
Ha tudni akarjuk, hogy egy futó program hogyan teljesít, a következő parancsot kell kiadnunk:

```
ps -o start,time,etime -p folyamatazonosító
```

Ekkor a `time` parancs – később még lesz róla szó – alapszintű kimenete jelenik meg, azzal a különbséggel, hogy nem kell megvárnunk a program futásának befejeződését.
A `ps` által átadott adatok túlnyomó része a */proc* fájlrendszeren keresztül is elérhető, de ha parancsfájlt írunk, a `ps` használatával jobban hordozható kódot kapunk. Soha nem tudhatjuk, hogy a rendszermag egy komolyabb módosítása miatt mikor válik működésképtelenné a */proc* fájlrendszerben kutakodó parancsfájlunk. Maradjunk inkább a `ps`-nél.

5. time

A `time` parancs segítségével leginkább kódunk teljesítményét tudjuk vizsgálni. Alapszintű kimenetében a tényleges, a felhasználói és a rendszeridő szerepel. A tényleges idő az az időmennyiség, amely a kód elindulásának és kilépésének időpontja között telt el. A felhasználói és a rendszeridő rendre a felhasználó által a kód és a rendszermag futtatására fordított idő.
A `time` parancs kétféle változatban létezik. A héj tartalmaz egy beépített változatot, amely csak ütemezési adatokat szolgáltat. A */usr/bin* könyvtárban található változat több adatot ad, és a kimenet formázását is lehetővé teszi. Ha a beépített helyett ez utóbbit akarjuk használni, akkor a parancs kiadásakor egy visszafelé hajló perjelet kell annak elejére írnunk, ahogy az a példákban is látható.
A Linux ütemezőjének alapszintű ismerete jelentősen megkönnyíti a kimenet értelmezését, ám ha éppen az ütemező működését akarjuk tanulmányozni, akkor ezzel az eszközzel ezt is megtehetjük. A valós idő például jellemzően na-

© Kiskapu Kft. Minden jog fenntartva

© Kiskapu Kft. Minden jog fenntartva

gyobb, mint a felhasználói és a rendszeridő összege. Ennek oka az, hogy a rendszerhívásokban blokkolva eltelt idő nem számít bele a folyamat futási idejébe, ilyenkor ugyanis az ütemezőnek megvan a lehetősége arra, hogy más folyamatoknak adja az erőforrásokat. Az alábbi példában a `sleep` parancs futása egy másodpercig tart, de mérhető rendszer- vagy felhasználói időt nem vesz igénybe:

```
\time -p sleep 1
real 1.03
user 0.00
sys 0.00
```

A következő példával azt szemléltetem, hogy egy folyamat hogyan töltheti teljes futási idejét a felhasználói térben. Ebben az esetben a *Perl* a `log()` függvényt hívja meg hurkolva, ennek futásához a rendszermagra gyakorlatilag nincs szükség:

```
\time perl -e ´log(2.0) foreach(0..0x100000)´
real 0.40
user 0.20
sys 0.00
```

További példaként lássunk egy nagy mennyiségű memóriát lekötő folyamatot:

```
\time perl -e ´$x = ´a´ x 0x1000000´

0.06 user 0.12 system 0:00.22 elapsed 81% CPU
(0 avgtext + 0 avgdata 0 maxresident) k
0 inputs + 0 outputs (309 major+8235 minor)
pagefaults
0 swaps
```

Ebben az esetben a `pagefaults` érték, vagyis a laphibák száma érdekel bennünket. Noha a GNU `time` parancs rengeteg adatot tesz elérhetővé, a 2.4-es sorozatú *Linux* rendszermagok csak a fő- és az allaphibákat követik figyelemmel. Főlaphiba az, amelynél szükség van be/kivitelre, allaphiba pedig az, amelyiknél nincs.

7. nm

Segítségével a megadott futtatható vagy objektumfájlon belüli szimbólumnevekről gyűjthetünk adatokat. Alapesetben a kimenetben egy szimbólumnév és annak képzetes címe jelenik meg. Hogy mindez mire jó? Tegyük fel, hogy éppen kódot fordítunk, és a fordító hibát jelez, miszerint a `_pelda` szimbólum feloldatlan. Átfésüljük a kódunkat, de nem jövünk rá, hogy hol használjuk ezt a szimbólumot. Lehet, hogy valamelyik sablonból származik, esetleg a kóddal együtt lefordított beemelt fájlok sokaságának egyikében szereplő makróról van szó. A kiadandó parancs:

```
nm -guA *.o | grep pelda
```

Ezzel minden a `pelda` szimbólumra hivatkozó modult megkapunk. Ha azt akarjuk tudni, melyik könyvtárban található a `pelda` megadása, a következő parancsot adjuk ki:

```
nm -gA /usr/lib/* | grep pelda
```

Az `nm` parancs a *C++* nevek kibontására is alkalmas, ami főleg *C* és *C++* kód keverésekor jöhet jól. Például, ha egy *C* függvényt `extern` „C” nélkül adunk meg, akkor a következőhöz hasonló fordítási hibát kapunk:

```
undefined reference to `cfunc(char*)´
```

Egy nagyobb méretű, szegényesen megadott fejlécekkel ellátott tervezetnél nem kis feladat a hibás modult megkeresni. Ha az összes feloldatlan szimbólumra rá szeretnénk keresni az objektumfájlokban (a nevek kibontásával), a következő parancsot állítsuk össze:

```
nm -guC *.o
extern-c.o:cfunc
no-extern-c.o:cfunc(char*)
```

Az első modul rendben van, a második viszont nem.

7. strings

A `strings` segítségével bináris fájlokba ágyazott ASCII karakterláncokat tudunk keresni. Tulajdonképpen jóra és rosszra egyaránt alkalmas. A jó oldal: kinyomozhatjuk, hogy az szabványos kimenetre kerülő rejtélyes szöveg melyik könyvtárból származik:

```
strings -f /usr/lib/lib* | grep “rejtélyes üzenet”
```

A rossz oldal: a karakterláncok alapján ki lehet deríteni, hogy milyen formátumot meghatározó karakterláncokat használunk, amivel akár sebezhető pontokat is lehet találni programunkban. Ez az oka annak, hogy a programokba soha nem szabad felhasználói neveket és jelszavakat beépíteni. Ugyanezért nem rossz ötlet saját programunkat is megvizsgálni vele, így legalább tudjuk, hogy egy rafináltabb programozó mennyire lát bele a kártyáinkba. A GNU ***binutils*** részeként elérhető `strings`-változat számtalan hasznos lehetőséget biztosít.

8. od, xxd

Ez a két parancs lényegében ugyanazt csinálja, ám némileg eltérő szolgáltatásokat nyújtanak. Az `od` bináris fájlokat tud tetszőleges formátumra alakítani. Ha olyan programokkal dolgozunk, amelyek nyers bináris fájlokat állítanak elő, az `od` rendkívül nagy segítséget jelenthet. Bár neve az ***octal dump***, az oktális kiíratás kifejezésből származik, az adatokat decimális és hexadecimális formátumba is képes kiírni. Az `od` egészeket, IEEE lebegőpontos értékeket és egyszerű bájtokat tud kezelni. Több bájtos egész vagy lebegőpontos értékeknél a futtató gép bájtsorrendjétől függ a kimenet.

Az `xxd` szintén bináris fájlokat írat ki, de nem próbálja egészekként vagy lebegőpontos értékekként értelmezni azokat. Így a futtató gép bájtsorrendje nem befolyásolja a kimenetet, ami a fájl jellegétől függően zavaró és előnyös is lehet. Példaként hozzunk létre egy négybájtos fájlt egy Intel alapú gépen:

```
$ echo -n abcd > pelda.bin
$ od -tx4 pelda.bin
0000000 64636261
0000004
```

```
$ xxd -g4 pelda.bin
0000000: 61626364                  abcd
```

Az od kimenete egy bájtfelcserélt 32 bites egész, az xxd-é viszont egy négy bájtból álló csoport, amelynek bájtsorrendje a fájléval megegyező. Ha az abcd karakterláncot akarjuk megkeresni, az xxd-t kell használnunk, ha viszont a 0x64636261 32 bites számot szeretnénk fellelni, akkor az od-ot vegyük elő.
Az xxd a kimenetet binárisan is képes formázni, valamint bináris fájlt képes C tömbbé alakítani. Tegyük fel, van egy bináris fájlunk, amit egy C program egy tömbjébe szeretnénk elhelyezni. Ezt egyetlen módon tehetjük meg: az alábbi módon létrehozunk egy szöveges fájlt.

```
$ xxd -i pelda.bin

unsigned char pelda_bin[] = {
  0x61, 0x62, 0x63, 0x64
};

unsigned int pelda_bin_hossz = 4;
```

Ha olyan programokkal dolgozunk, amelyek nyers bináris fájlokat állítanak elő, az od rendkívül nagy segítséget jelenthet.

9. file

UNIX és ***Linux*** alatt a fájlnevek kiterjesztésére vonatkozóan soha nem volt kötelező előírás. A névadási szokások ugyan fejlődtek, de csak irányelvekről és nem kötelezően betartandó előírásokról van szó. Ha egy digitális képet ***kep00.exe*** névvel akarunk ellátni, megtehetjük. Linuxos fényképkezelő alkalmazásunk gond nélkül fogadni fogja a fájlt, függetlenül annak nevétől, amit nekünk viszont nehezebb lesz megjegyeznünk.
A file parancs például akkor lehet segítségünkre, ha egy összeomlott webböngészőből kell kibányásznunk egy összezavart nevű fájlt. Tegyük fel például, hogy a fájl nevének ***pelda.proba.hello.vilag.tar.gz*** kellene lennie, a valóságban viszont ***pelda.proba***. A kiadandó file parancs a következőképpen alakul:

```
$ file pelda.proba

pelda.proba: gzip compressed data, was
“pelda.proba.hello.vilag.tar”, from Unix
```

Előfordulhat, hogy kapunk egy terjesztést, aminek a ***bin*** könyvtárában tucatnyi fájl található, futtatható fájlok és parancsfájlok vegyesen. Tegyük fel, hogy ki szeretnénk válogatni a héjprogramokat. Próbálkozzunk ezzel:

```
$ file /usr/sbin/*  | grep script

/usr/sbin/ezmegmi:  a /bin/bash script text
                    executable

/usr/sbin/xconv.pl:     a /usr/bin/perl script
                        text executable
```

A file parancs a ***bin*** könyvtár összes fájlját megvizsgálja, a grep pedig kiszűri azokat, amelyek nem parancsfájlok. Néhány további példa:

```
file core.4867

core.4867: ELF 32-bit LSB core file Intel 80386,
➥version 1 (SYSV), SVR4-style, from ´abort´

file /boot/initrd-2.4.20-6.img

/boot/initrd-2.4.20-6.img: gzip compressed data,
➥from Unix, max compression

file -z /boot/initrd-2.4.20-6.img

/boot/initrd-2.4.20-6.img: Linux rev 1.0 ext2
➥filesystem data (gzip compressed data, from
➥Unix, max compression)
```

Ne feledjük, ahogy a könyvek tartalmát sem ítélhetjük meg borítójuk szerint, úgy a fájlok tartalmát sem mérhetjük fel csupán nevük alapján.

10. objdump

Ez egy különlegesebb eszköz, nem kifejezetten kezdők számára. Egyfajta objektumfájlokhoz készült adatbányász programnak tekinthető.
Az objektumkód rengeteg értékes adatot tartalmaz, az objdump futtatásával ezeket tudjuk elérni. Segítségével például kiírathatjuk a forrássorokba kevert assembly kódokat; ez az, amit a gcc -S, ki tudja miért, de nem tesz meg. Működéséhez az objektumkódot a debug (-g) kapcsolóval kell lefordítani:
objdump –demangle –source objektum.o

Az objdump-pal elhalálozás utáni hibaelemzés céljából mag (core) fájlból is kinyerhetünk bináris adatokat, ha hibakereső program éppen nem áll rendelkezésünkre. Teljes példát most helyszűke miatt nem ismertetnék, de annyit elmondanék, hogy az nm vagy az obdump segítségével először meg kell határoznunk a képzetes címet, ezután az objdump -x paranccsal az összes képzetes címhez tartozó fájleltolást ki tudjuk gyűjteni. Az utolsó lépés az objdump ismételt futtatása, ez ugyanis nem ***ELF*** fájlformátumokból is képes olvasni, ellentétben a ***gdb***-vel és az egyéb eszközökkel.
Írásom nem annyira útmutatóként, inkább kiindulópontként szolgálhat termelékenységünk növeléséhez. Az említett parancsok mindegyikéhez kimerítő leírás tartozik a linuxos ***man*** és ***info*** oldalak között. További tájékoztatást és ötleteket ezekben érdemes keresni.

Linux Journal 2004. szeptember, 125. szám

John Fusco programfejlesztő a General Electric Healthcare-nél (korábban GE Medical Systems), ahol linuxos programokat és illesztőprogramokat tervez a GE Lightspeed sorozatú számítógépes tomográfjaihoz (www.gemedicalsystems.com/rad/ct/products/light_series/index.html).

© Kiskapu Kft. Minden jog fenntartva

© Kiskapu Kft. Minden jog fenntartva

GRUB rendszertöltő – az alapoktól kezdve (2. rész)

Telepítés, testreszabás, finomhangolás.

Előző cikkünkben megismerkedtünk a *GRUB* rendszertöltő felépítésével, filozófiájával, alapjaival. Eljött az ideje annak, hogy a tettek mezejére lépjünk, s kipróbáljuk a gyakorlatban is, mire képes a program. Ehhez először megpróbáljuk telepíteni a gépünkre a *GRUB*-ot, s ha ez sikeresen megtörtént, igényeink szerint testreszabjuk a rendszert. Egyszóval ebben a cikkben átállunk *LILO*-ról *GRUB*-ra. Látni fogjuk, hogy a dolog egyszerűbb, mintha *LILO*-t konfigurálnánk, s cserébe mégis egy jóval kényelmesebb eszközt kapunk.

Amire szükségünk lesz

Mielőtt mindehhez hozzákezdenénk, nem árt tisztázni, hogy milyen környezetre van szükségünk. Az alább bemutatott fogások ugyanis egy olyan gépet igényelnek, amelyen egynél több operációs rendszert használunk. Természetesen egyetlen Linux is bőven elég a feladat elvégzéséhez, de ekkor nem fogjuk látni, hogy milyen előnyökkel jár a *GRUB* használata. (Megjegyzem: egyetlen operációs rendszer használata esetén is előnyösebb a *GRUB*, mint a *LILO*.)
Ha tehát szép színesben szeretnénk látni a dolgokat, nem árt, ha van egymás mellett legalább egy Windows és egy Linux operációs rendszer, és mindkettő működőképes.
Ezen túl szükségünk van a *GRUB* csomagjaira, amelyek ma már a legtöbb terjesztésben helyet kapnak, így nemhogy fordítanunk nem kell, még csak utánajárni sem kell a szükséges elemeknek. Egyszerűen rendszerünk csomagkezelőjével telepíthetjük a kívánt összetevőket. (Ezek neve Debian, ill. SuSE alatt *grub* és *grub-doc*.). A csomagok telepítése nem jár együtt a rendszertöltő telepítésével, ami azt jelenti, hogy pusztán a telepítéstől nem törlődik a régi MBR (Master Boot Record). Nem kell tehát attól félni, hogy a gépünk esetleg nem indul el. A régi rendszertöltő egészen addig a helyén marad, amíg az újat be nem állítjuk.
A telepítést megelőzően igencsak ajánlatos biztonsági indítólemezt készítenünk, amivel az esetleges „katasztrófák" esetén újraindíthatjuk a rendszert – akár a régi rendszertöltő visszaállításának, akár egy új telepítési kísérlet céljából.

Nosza, telepítsünk!

Mint már említettem, a rendszer csomagkezelőjének segítségével telepítsük a gépünkre a *GRUB* programfájljait. Ha valamilyen oknál fogva ez nem megoldható, vagy újabbat szeretnénk használni, mint ami rendelkezésre áll, akkor látogassunk el a GRUB honlapjára (➲ http://www.gnu.org/software/grub/) és a letöltés után fordítsuk le, majd telepítsük. Ha ezzel megvagyunk, be kell írnunk a *GRUB* első szakaszát a merevlemez fő rendszerindító területére (Master Boot Record – MBR), illetve azt a bizonyos „másfeledik" szakaszt az azt követő területre, ám ez utóbbi már automatikusan zajlik. Kétféle módon telepíthetjük a *GRUB*-ot az MBR-be, ezek közül azonban számunkra most csak az egyik érdekes. Létezik egy parancsállomány (script), amely `grub-install` névre hallgat. Ezt kell a megfelelő paraméterekkel meghívni, s a művelettel gyakorlatilag készen is vagyunk.
A parancsállomány átmásolja a lenyomatok könyvtárából (image directory) a megfelelő állományokat az indítókönyvtárba (boot directory), amely alapértelmezésként a */**boot**/**grub***.
Ezután meghívja a *GRUB*-ot magát, hogy a kapcsolóknak megfelelően írja be az egyes alkotórészeket a merevlemez megfelelő területeire. Lássunk erre egy példát.
Minden valószínűség szerint a BIOS szerinti legelső lemezre szeretnénk telepíteni a *GRUB*-ot, és ajánlatos úgy eljárni, hogy ez valóban – IDE eszköz esetén – az elsődleges mester (primary master) merevlemez legyen. Ha ez nem így van, és másik merevlemezre telepítjük a *GRUB*-ot, akkor láncolt betöltéssel (chainloading – lásd Linuxvilág magazin 43. szám, 62-63. oldal) kell az elsődleges lemezen található rendszertöltővel elérni a *GRUB*-ot. Megjegyzem, erre általános használat esetén igen kicsi az esély, ezért maradjunk most az előző az eshetőségnél.
Nincs más dolgunk, mint kiadni az alábbi parancsot:

```
grub-install /dev/hda
```

Ez természetesen független attól, hogy az aktuális Linux partíciónk hol van – ezt a telepítő parancsállomány majd elintézi. A telepítő alapértelmezetten a */**boot**/**grub*** könyvtárban keresi a rendszerállományokat, ám ha a gyökérben is találna egyet, akkor azt használná. Erre csak különleges esetben kell vigyáznunk, akkor, ha valamiért a gyökérbe másoltuk a *GRUB* szakaszait képező lenyomatokat. Ha ez így volna, akkor a `–root-directory=<elérési út>` kapcsoló megadásával segíthetünk a problémán, amely kapcsoló természetesen akkor is hasznos lehet, ha más könyvtárban találhatók a *GRUB* állományai – de mint említettem, ez általános esetben nem fordulhat elő.
A parancs kiadását követően pár másodperces ellenőrzés és telepítés következik. Ha ez sikeresen megtörténik, akkor elvileg minden rendben, készen vagyunk. Előfordulhat azon-

ban, hogy figyelmeztetést kapunk. Amíg ez valóban csak figyelmeztetés, addig nem valószínű, hogy gondunk lesz belőle. Én is rendre kapok ilyeneket, pedig mindössze egyetlen merevlemez van a gépemben. Ennek ellenére még mindig remekül működött a dolog.

Hogyan tovább?

Remek! A rendszertöltő most már a helyén van, tenné is a dolgát, ha most újraindítanánk a gépet. Ezt most inkább ne tegyük, előbb készítsünk egy menüt, hogy kényelmesebben választhassunk a telepített rendszerek között. A menüt a *GRUB* alapértelmezetten a */boot/grub/menu.lst* nevű állomány alapján állítja elő, tehát itt lehet megadni az egyes menüpontokat, s ezen kívül még a menü színét, az esetleges háttérképet illetve grafikus beállításokat. Ugyanakkor ne felejtsük el, hogy ezeknek a lehetőségeknek a megvalósítása még erősen gyerekcipőben jár. Jómagam még színeket sem használok, teljesen jól működik az egyszerű, fekete alapon fehér betűs változat is.

Hogy ne kelljen sokat gyötrődnünk a menüvel, a *GRUB* fejlesztői, illetve a leírás készítői mellékeltek egy példaállományt, melyet a csomagok dokumentációi között találunk. Ez a legelterjedtebb terjesztésekben a */usr/share/doc* könyvtár, s ezen belül keressük a *grub/examples/menu.lst* fájlt, amit másoljunk át valahová, ahol kedvünkre alakíthatjuk.

A menu.lst fájl

Ha szerkeszteni kezdjük a fájlt, észrevehetjük, hogy a beállítások két részre oszthatók: vannak általános beállításokra, illetve magukra a menüpontokra. Ezeken kívül természetesen még számos egyéb, haladó szintű előzetes beállítás is létezik.

Az általános beállítások a menü automatikus kezelésére vonatkoznak. Nézzük sorban a lehetőségeket.

`timeout <másodperc>`: Meghatározza, hogy mennyi ideig várakozik a rendszertöltő a felhasználói beavatkozásra, mielőtt az alapértelmezett operációs rendszer automatikusan betöltődne. Ha akármelyik billentyűt lenyomjuk, akkor ez a várakozási idő érvényét veszti. Ez azt jelenti, hogy ezután addig nem indul el semmi, amíg magunk nem választunk ki egyet a lehetőségek közül. Az alapértelmezett rendszer indításáig hátralévő időt a gép indításakor a rendszertöltő menü jobb alsó sarkában olvashatjuk.

`default <menü száma>`: Ez határozza meg, hogy a `timeout` pontban megadott idő eltelte után melyik menüpontot töltse be automatikusan a *GRUB*. A menüpontoknak nincs külön száma, a menüfájlban elhelyezett sorrendjük szerint sorszámozódnak, de nullától kezdődően.

`fallback <menü száma>`: Abban az esetben, ha az alapértelmezett menüpont nem működne, az itt megadott számú menüpontra ugrik a *GRUB*, s ezt tölti be. E menü számára is ugyanazok a szabályok érvényesek.

`hiddenmenu`: Ha ezt beírjuk a menüfájlba, rendszertöltéskor a menü nem jelenik meg, s a lejárati idő (`timeout`) eltelte után az alapértelmezett (`default`) menüpont aktiválódik. Ha ezen alkalomadtán változtatni szeretnénk, akkor az Esc billentyű megnyomásával előhozhatjuk a menüt.

Ezen kívül már csak egy menüparancs van, ez pedig a `title`. Ezzel készíthetünk új menüpontot az alábbi módon:

```
title <menüpont látható neve amit majd
➥kiválasztunk>
```

Létrehozza a menüpontot. Ez alá kell sorban megadni azokat a parancsokat, amelyeket a menüpont kiválasztásakor szeretnénk végrehajtani. Ezeknek nem kell feltétlenül a rendszertöltésre vonatkozniuk. Választhatunk színt, megadhatunk hálózati beállításokat, és így tovább.

A *GRUB* a menüpont aktiválásakor a `title` után a fájl végéig, vagy a következő `title` utasításig szereplő minden parancsot végrehajt. Ezek a bizonyos parancsok valójában a GRUB parancsértelmezőben kiadható utasítások, amelyek egy héjprogramhoz hasonlóan hajtódnak végre. A parancsok közül most csak a legfontosabbakat nézzük meg.

Linux rendszer betöltése menüből

Ez az menübejegyzés az alábbi módon néz ki:

```
title Linux betoltese
root (hd0,1)
kernel /boot/2.6.7/bzImage root=/dev/hda2
```

A `root` parancs segítségével jelölhetjük ki a gyökérpartíciónkat, amin azután a rendszermag lenyomatot keresni fogjuk. A lemez megadása két azonosítóval történik. Az első azonosító a lemez BIOS által ismert sorszámát mutatja. Ez a sorszámozás nullával kezdődik. A `hdd` lemez tehát `hd3`-mal azonosítható a `GRUB` ezen parancsa számára. A második szám a lemezrész sorszáma, szintén nullától kezdődően. Esetünkben a `hda2`-n van a Linux, amelyet az 1-es szám képvisel. Mindezek megállapításához természetesen alaposan ismernünk kell a rendszerünket.

A `kernel` paranccsal állíthatjuk be, hogy hol is található magának a rendszermagnak a lenyomata, majd ezt követően szóközökkel elválasztva egymás után adhatjuk meg a rendszermag kapcsolóit. Ezek közül egy mindenképp kötelező: a `root` paraméter, amely nem azonos az előző sorban szereplő *GRUB* utasítással. A rendszermagnak ez azt jelzi, hogy melyik lemezrészt kell a Linux gyökerekének tekintenie. Néhány terjesztés használ még `initrd` állományt is. Ha a mi Linuxunk is ilyen, az állományt `initrd <elérési út>` paranccsal adhatjuk meg.

© Kiskapu Kft. Minden jog fenntartva

© Kiskapu Kft. Minden jog fenntartva

Windows rendszer betöltése menüből

Ez az menübejegyzés az alábbi módon néz ki:

```
title Barmilyen Windows inditasa
rootnoverify (hd0,0)
makeactive
chainloader  +1
```

A `rootnoverify` parancs a `root` parancs egy különleges változata, amely a parancs kiadásakor nem ellenőrzi a lemezrész meglétét, állapotát, és egyéb paramétereit. Egyszerűen elhiszi nekünk, hogy ott van. Abban az esetben használjuk, amikor a *GRUB* számára ismeretlen típusú lemezrészt, és ezzel együtt ismeretlen típusú operációs rendszer helyét adjuk meg. A kapcsolókat tekintve ugyanúgy kell használni, mint az egyszerű `root` utasítás.
A `makeactive` parancs a *DOS* szempontjából aktívnak jelöli meg az előzőleg megadott lemezrészt. Mint tudjuk, a *Microsoft* termékei csak úgy hajlandók (az esetek jelentős többségében) elindulni, ha aktívnak kijelölt lemezrészről töltjük a rendszert. Ezt biztosítja ez a bizonyos parancs.
A `chainloader +1` parancs kiadásával a láncolt betöltést aktiváljuk, melynek hatására a betöltés vezérlése átadódik az adott lemezrész rendszertöltőjéhez. Ez utóbbi természetesen úgy érzi majd, mintha a BIOS ébresztette volna fel, s szó nélkül elindul a kiválasztott operációs rendszer.
Ha készen vagyunk a menüvel, akkor másoljuk át a */grub/boot* könyvtárba (*menu.lst* néven), s a következő indítás során már az új menübejegyzés szerinti rendszertöltő indul, próbáljuk ki hát!

GRUB indítólemez készítése

Indítólemez készítésére ismét több lehetőségünk is van. Az első a *Linux* `dd` parancsa, amellyel átmásolhatjuk a *GRUB* szakaszait egy hajlékonylemezre. A másik módszernél magát a *GRUB*-ot hívjuk meg, amely a merevlemez helyett a meghajtóban lévő hajlékonylemezre települ. Számunkra az első lehetőség az érdekes. Ehhez lépjünk be a *GRUB* lenyomat-könyvtárába (*/usr/lib/grub/i386-pc*), majd adjuk ki a következő parancsokat, egymás után.

```
dd if=stage1 of=/dev/fd0 bs=512 count=1
dd if=stage2 of=/dev/fd0 bs=512 seek=1
```

Rendszerindítás GRUB indítólemez használatával

Előfordulhat (nem csak most, a telepítés következtében, hanem más esetekben is), hogy valami miatt csak indítólemez használatával tudjuk életre lehelni rendszerünket. Ha például újratelepítjük Windowst, akkor az rossz esetben felülírta a *GRUB*-nak az MBR-be telepített részét. Ha ez után szeretnénk a *Linuxot* elindítani, akkor egy *GRUB* lemezzel kell újra életre kelteni pingvinünket. Igaz ugyan, hogy most még semmi bajunk nincs, de amolyan tűzoltási gyakorlatként azért próbáljuk ki a lemezes módszert. Ha floppy-ról indítjuk a gépet, azonnal a *GRUB* már emlegetett parancsértelmezőjét kapjuk a megszokott menü helyett. Ebben ugyanazokat a parancsokat használhatjuk, mint a menüfájlban. Próbaképp indítsuk el a *Linuxot* ebből a parancssorból. Ehhez adjuk ki egymás után ugyanazokat a parancsokat, melyeket a Linux rendszer betöltése során is használnánk.

Minden egyes parancs beírásakor TAB-bal kiegészíthetjük a beírandó parancsot, kapcsolót, illetve listát is kapunk a lehetséges értékekről, csakúgy, mintha a bash-ben tevékenykednénk. Még az egyes lemezrészek fájlrendszerét is látjuk – feltéve, hogy az megegyezik a telepített *GRUB* által ismert lemezrész formátumával. Ellenkező esetben csak a lemezrészen található fájlrendszer típusáról kaphatunk információt.
A parancsok kiadása után nem történik semmi. Ahhoz, hogy elindíthassuk a megadott operációs rendszert, adjuk ki a `boot` parancsot, amely az eddig begépeltek alapján elindítja a rendszertöltést. Ha betöltődött a rendszer, máris számtalan lehetőségünk van a helyreállításra, telepítésre vagy amire éppen szükségünk van.
Most pedig nézzük meg, hogy sérült MBR esetén hogyan állíthatjuk helyre a GRUB-ot, hasonlóan, indítólemez segítségével. Ez egyébként nem más, mint a már említett második telepítési forma. Lássuk, mely utasításokat kell kiadnunk.
Először is a `root` parancs segítségével adjuk meg, melyik lemez hányadik lemezrészén található a *GRUB* programfájljainak telepített változata. Ez alapértelmezetten a linuxos lemezrészünk, tehát az, amit a példánk során a menüben is megadtunk (`hd0,1`). Ezt követően adjuk ki a `setup (hd0)` parancsot, amely az elsődleges lemez fő lemeztöltő területére helyezi a *GRUB* első szakaszát, s gondoskodik az esetleges „másfeledik” szakaszról.
Ha nem vagyunk biztosak abban, hogy melyik lemezrészen van a *GRUB*, adjuk ki a `find /boot/grub/stage1` utasítást, mely megkeresi azt a lemezrészt, melynek */boot/grub* könyvtárában ott van a *GRUB* első szakasza, majd kiírja a lemezrész nevét. És ezzel készen is vagyunk, helyreállítottuk a *GRUB*-ot.
Bármely indítás során elérhetjük ezt a bizonyos – rendkívül hasznos – parancssort. Indításkor a menünél csak nyomjuk meg a C billentyűt, és máris ott találjuk magunkat. Ezek után egy `help` parancs segítségével kilistázhatjuk, hogy milyen elérhető utasítások közül válogathatunk. Látni fogjuk, hogy a rendszertöltéssel kapcsolatban ezzel gyakorlatilag bármilyen fontos feladatot elvégezhetünk. Ha vissza szeretnénk térni a menühöz, csak nyomjuk le az ESC billentyűt.

Végezetül

Láthattuk, hogy a *GRUB* rendszertöltő esetében messze több lehetőségünk van a működés beállítására mint bármely hasonló program esetében. Ugyanakkor határozottan állíthatom, hogy ennek ellenére használata jóval egyszerűbb, mint a *LILO*-é. Tekintve, hogy a telepítésen túl számos egyéb, hasznos fogásra is kitértem, őszintén remélem, hogy azok számára is értékes információval szolgáltak ezek a cikkek, akik olyan terjesztést használnak, amely már eleve a *GRUB*-ot telepíti. Ilyen például a *SuSE*, de tudomásom szerint az összes friss *Linux* változat a *GRUB*-ot részesíti előnyben. Talán nem véletlenül.

Komáromi Zoltán
(komi@kiskapu.hu)
23 éves, a BME hallgatója, mellette PHP-programozóként dolgozik. Kedvenc területe a multimédia.

© Kiskapu Kft. Minden jog fenntartva

Haladjunk a GIMP-pel

Mára már elég világossá vált számomra, hogy a GIMP fejlődését nekem sem szabad figyelmen kívül hagynom, így érdemes szólni néhány szót arról, hogy milyen változások történtek a 2.0-ás verzióban.

Elsősorban a kód belső felépítésében történtek változtatások, mondhatni, hogy egy teljesen új programmal találkozhatunk. Elméletben lehetőség van ***Python*** beépülő modulok használatára is, azonban ezt a lehetőséget többszöri újrafordítás után sem volt alkalmam kipróbálni. Fontos megemlíteni, hogy a ***GIMP*** programfejlesztésének alapja – a ***GTK*** – egy kicsit gyorsabb tempót diktált a fejlődésnek, így a programozók is alkalmazkodtak az új helyzethez. Arról van szó, hogy a ***GTK*** elméletileg azért jött létre, hogy a grafikus elemek megjelenítését a ***GIMP***-ben megvalósítsa, de maga a program még csak ebben a verzióban kezdi el használni a ***GTK+ 2.2***-es verzióját. További, kevésbé látványos változásokat találhatunk a belső felépítésben. Tisztább, gyorsabb lett a ***GIMP*** motorját adó ***libgimp*** rutinkönyvtár kódja, új rutinok kezelik az előnézeti képeket, és a program alkalmas 64 bites környezetben való futtatásra és természetesen képes használni processzorunk SIMD utasításkészletét (MMX, MMXEXT, stb) is.

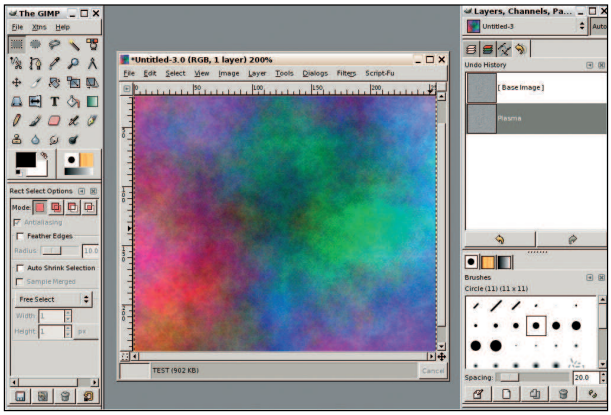
A látható újdonságok

A belső, alig látható változások után az egyik meglepetés a ***GIMP 2.0*** elindítása után érhet bennünket. Az eddig megszokott sok-sok ablak helyett, az egyes eszközkészleteket és műveleteket a fejlesztők most már jobban csoportosították, ebből adódóan a felhasználói felület nagymértékben átalakult. A különféle beállítások, ecsetek, színválasztó gombok az általános eszközkészlettel egy ablakban jelennek meg, és a kis ablakok leválaszthatók hordozójukról.

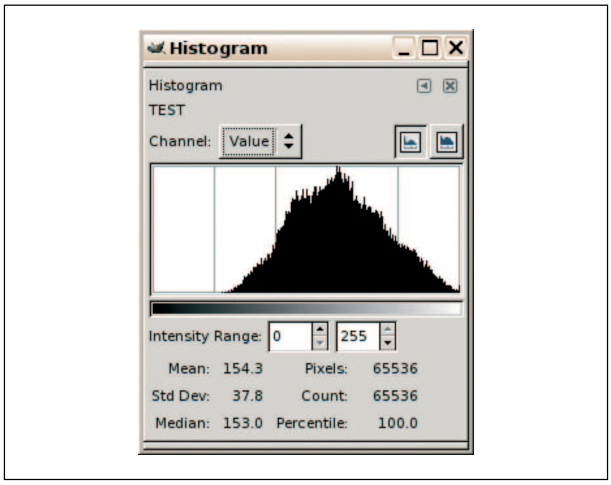
Ha megnyitunk egy képet, annak ablakában rögtön feltűnhet, hogy a menü eléréséhez már nem kell feltétlenül a jobb egérgombot használni, hiszen minden képablakban megtalálható a menüsor. Szokás kérdése talán, de ez nem biztos, hogy meggyorsítja a munkát. Azonban nem kell elkeseredünk, az eddigi jobb-gombos megoldás is működik.

A kisebb változások, újdonságok használat közben kerülnek elő, viszont ebben a sorozatban már nem térhetek ki újra minden eddig tárgyalt területre. Azonban fontosnak tartom tehát néhány érdekességre felhívni a figyelmet.

Amint az a 2-es képen látható, a hisztogram megjelenítése már külön párbeszédablakba került. Az eddig bemutatott szűrők és átalakítások között is találunk újabbakat. Az ***Általános*** (***Generic***) szűrők között található az ***Erode*** és a ***Dilate***. Az ***Erode*** alkalmas arra, hogy a képeinken a különálló területeket úgymond vékonyítsuk. Akár körvonalról vagy kitöltött területről van szó, az ***Erode*** szűrő alkalmazása után a körvonalak vékonyabbak lesznek és a kitöltött területek szélén is láthatjuk a szűrő hatását.

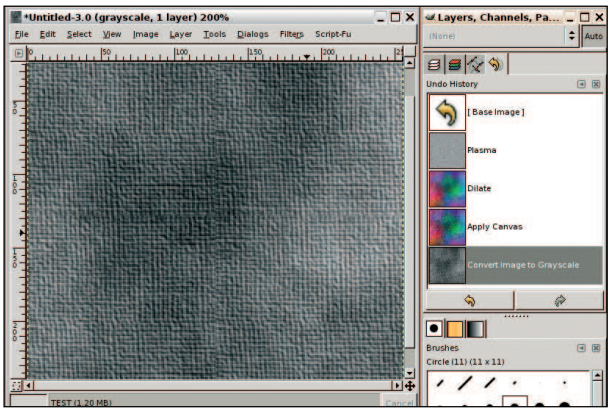


1. kép A GIMP új felülete

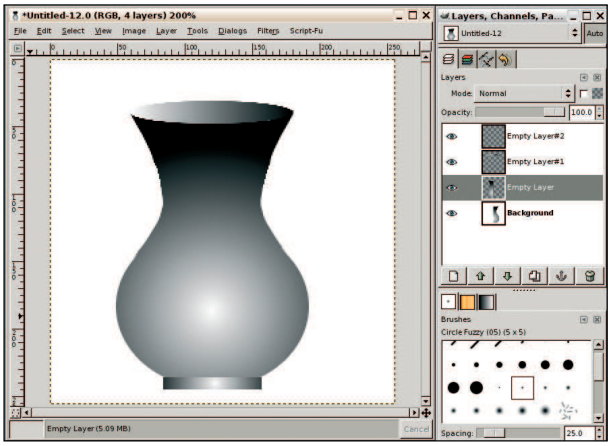


2. kép A hisztogram ablaka

© Kiskapu Kft. Minden jog fenntartva



3. kép Műveletek visszavonása



4. kép A váza rétegei

A ***Dilate*** szűrővel pontosan az előbbivel ellenkező hatást érhetjük el, a körvonalak és a területek kicsit „kövérebbnek" látszanak alkalmazása után.

A következő szembetűnő új lehetőség az, hogy a ***GIMP*** rögzíti a műveleteket és így nyomon követhetjük a képek átalakításának menetét. Természetesen az új eszköz nem csak nyomkövetésre alkalmas, hiszen az átalakításokat tetszőleges mélységben vissza is vonhatjuk. A 3-as kép jobb oldalát szemlélve könnyebben alkothatunk képet a ***GIMP*** fejlesztői által ***Undo history***-nak nevezett eszközről és annak gyakorlati hasznáról.

A fő eszköztárban is találhatunk új gombokat, azonban ezek nem mindegyike új, egyszerűen csak a korábban egybetartozó eszközök különválasztásáról van szó. Az új ***GIMP***-ben ahhoz, hogy egy képet elforgassunk, már nem kell különböző eszközbeállításokon keresztülhaladni, hanem csak egyszerűen kiválasztjuk az elforgatásra szolgáló gombot, és elvégezzük a változtatást. Ugyanez érvényes az átméretezésre és a perspektíva korrekcióra is, melyek szintén külön gomb segítségével érhetők el.

Érdemes megemlíteni a kalligrafikus toll eszközt, aminek ikonja egy klasszikus töltőtollhoz hasonlít. A toll használatával rajzolhatunk olyan vonalakat és görbéket, melyek szélessége a rajzolás sebességétől függ. Ha lassabban húzzuk a vonalakat, akkor a tollból is több tinta folyik ki, a vonal vastagabb lesz, gyorsabb mozgás esetén pedig a toll hegye nem nyílik szét annyira, így a vonalunk is vékonyabb lesz.

Az áttérés után szintén egyszerűbbé válik az életünk, ha azonos színű területeket kell kiválasztanunk. A „***kiválasztás szín szerint***" lehetőség mindeddig a jobb gombos menüben több keresgélésre adott okot. Nagyon jó ötlet volt a fejlesztőktől, hogy külön gombra helyezték el ezt a gyakran használt menüpontot. Az eszköz ikonja egy háromszínű sáv feletti mutató kezet ábrázol.

A B billentyűvel elérhető egy másik nagyon hasznos eszköz, amelyet úgynevezett útvonalak létrehozására használhatunk. Az útvonalak például képezhetik kijelöléseink alapját vagy vastaggal kihúzhatjuk a körvonalukat, így különféle alakzatokat hozhatunk létre. Az igazi erősségük abban áll, hogy a rétegekhez hasonlóan egyszerre többet is tárolhatunk és a kijelölt területeinket ismételten kijelölhetjük. Az útvonalak kezelésére alkalmas beállításokat a rétegek kezelésére szolgáló ablakban, a harmadik fülecskén találjuk meg. A 3-as képen látható, vezérlőpontokkal megjelenített görbe ikonját kell keresnünk, az eszköztárban pedig szintén egy vezérlőpontokkal határolt görbe és egy töltőtoll képe található a megfelelő gombon.

Egy kis gyakorlás

Ismétlés és gyakorlás céljából készítsük el a négyes képen látható kezdetleges váza képét. Látható, hogy a váza szimmetrikus tárgy, tehát elég lesz az egyik felét elkészíteni.

A feladat nehézségét az adja, hogy a körvonal alakja nehezen bontható le az addig megismert primitívekre (kör, négyszög, ellipszis, stb.). Nem kell azonban megijedni, hiszen világosan látszik, hogy jól meghatározható szabályos görbékkel ki tudjuk alakítani a formát. Használjuk az előző eszközt, és hozzunk létre egy új útvonalat. Csak a váza külső vonalát kell megszerkeszteni, a belső – függőleges – vonalat a ***GIMP*** adja hozzá a kijelöléshez, amikor zárt területté alakítja az útvonalat. A vonal azonban csak akkor lesz függőleges, ha a görbe kezdő- és végpontjának vízszintes koordinátája megegyezik.

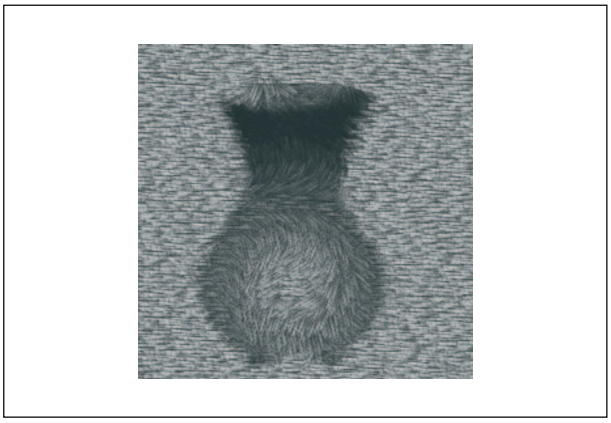
Az alapforma létrehozása után alakítsuk kijelöléssé az útvonalat, majd fessük ki színátmenettel. Az átmenet legyen sugaras, kezdőpontja a nagyjából a leendő legfényesebb pont legyen, végpontja pedig a váza szájának széle. Ezután hozzunk létre egy új réteget és rajzoljuk meg a váza talpának felét. Jelöljünk ki egy megfelelő méretű téglalap alakú területet és átmenettel fessük is ki.

A következő lépésben egyesítsük az előbbi két réteget (jobb egérgomb a rétegek kezelésénél, majd *'**Merge Down**'*) és jelöljünk ki mindent CTRL-A lenyomásával. A CTRL-C, CTRL-V billentyűkkel készítsünk másolatot róla, majd a vízszintes tükrözéssel (SHIFT-F) elkészíthetjük a váza másik felét. Ez helyezkedjen el egy új rétegen.

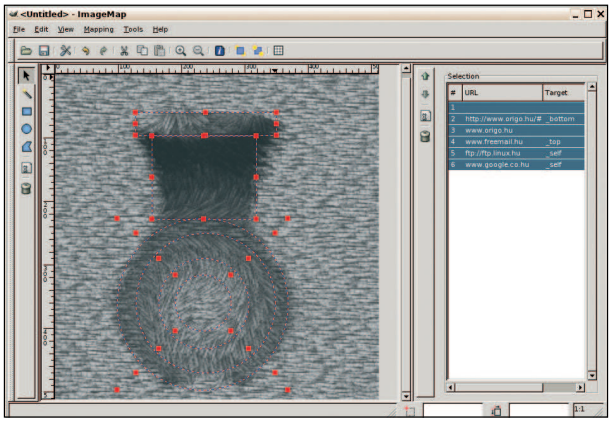
Hozzunk létre még egy réteget, majd jelöljük ki rajta a váza szájának helyét. Ezt is színátmenettel fogjuk kifesteni, az átmenet típusa lehet egyenletes (linear), hiszen ilyen kis területen nem szembetűnő a dőlés hiánya. Ezzel el is készítettük a vázát, már csak a különféle kiegészítő elemeket kell elhelyezni vagy átalakítani további felhasználás céljából.

Hatások és munkálatok

Az új lehetőségek megismerése és egy kis gyakorlás után kedves olvasóimnak bizonyára eszébe jut, hogy még igazából az előző változat képességeivel sincsenek teljes mérték-



5. kép Impresszionista váza



6. kép Image Map készítése a GIMP-pel

© Kiskapu Kft. Minden jog fenntartva

ben tisztában. Folytatjuk az ismerkedést a különféle szűrőkkel. A most bemutatásra kerülő szűrők közös tulajdonsága, hogy a szűrő elnevezés rájuk nézve nem igazán helytálló. Általában nem szűrnek ki semmit a képtartalomból, hanem inkább különféle hatásokat adnak hozzá, tehát jobban illik rájuk a „hatás" (Effect) elnevezés.

Varázsoljuk elő a jobb egérgomb segítségével a kép-menüt és válasszuk ki a ***Glass Effects*** menüpontot. Az új ***GIMP***-ben találhatunk egy már-már klasszikusnak számító nagyítólencse hatást, mégpedig a ***Apply Lens...*** menüpontban. Adjuk meg az üveg törésmutatóját és máris nagyítólencsén keresztül láthatjuk a képünket. A lencsék általában kör alakúak, de ez a tény nem akadályozza meg a programot abban, hogy az aktuális képarányoknak megfelelő ellipszis formájú nagyítást készítsen a képről.

A másik üveggel kapcsolatos hatás a ***Glass Tile...*** menüpontban, ugyanebben az almenüben található. A hatás lényege, hogy alkalmazása után a képet egy üvegablakon keresztül nézhetjük, amely nem teljesen sík felületű, hanem a megadott paramétereknek megfelelően (szélesség és magasság) apró darabokra tagolt tört üveg.

A különféle fényhatásokat a ***Light Effects*** menüben találjuk meg. A ***Flare FX*** a legegyszerűbb és hasonlít ahhoz, amikor egy fényforrás körül fényudvart látunk. Túl sok beállítási lehetőségünk nincsen, mindössze a hatás kiindulási koordinátáit kell megadnunk.

A ***Gflare*** már egy kicsit összetettebb hatások létrehozására alkalmas. Ez az a klasszikus lencsecsillanás, amikor a kamera optikájába a fény bizonyos szögből érkezik, és az optikán belüli fénytöréseket és csillanásokat látjuk a kamera által létrehozott képen. A kezdőpont meghatározásán kívül lehetőségünk van megadni a fényforrás sugarát, színeinek változását és azt is, hogy a csillanás milyen távolságra terjedjen ki és milyen szögből érkezzen a megfigyelő felé. Itt előre beállított értékeket találunk, de ha valami szépet alkotunk, akkor azt saját beállításként tárolhatjuk.

A ***Lighting Effects*** hatás segítségével létrehozhatunk különféle megvilágított felületeket, melyeken a kép elhelyezkedik. Meghatározhatjuk, hogy milyen fényforrással szeretnénk számolni, az hol helyezkedjék el a térben, milyen anyagra essék a fény és milyen legyen a kép felülete. Megadhatunk továbbá még egy másik képet, amelyen a környezetet ábrázoljuk és ha erősen fényvisszaverő anyagot határozunk meg, akkor a környezet is tükröződhet.

A ***Supernova*** hatás alkalmas különféle fényudvarok, csillagok vagy csillanások létrehozására vagy egy némi további munkával akár robbanást is megjeleníthetünk vele.

Az ***Artistic*** menüben találjuk a különféle hatásokat, melyekkel művészi kinézetet kölcsönözhetünk képeinknek. A legegyszerűbb ilyen hatás az ***Apply Canvas***. Alkalmazásához megválaszthatjuk a vászon szálainak haladási irányát és a szálak vastagságát, vagyis a vászon érdességét.

A ***Cubism*** menüpont választásával kubista hatást kölcsönözhetünk a képeinknek. Azt a kedves olvasóra bízom, hogy ezt a lehetőséget mikor használja. Véleményem szerint, ha valamit valahogyan elkészítünk, akkor az általunk válik egyedivé, nem mások szolgai utánzásától.

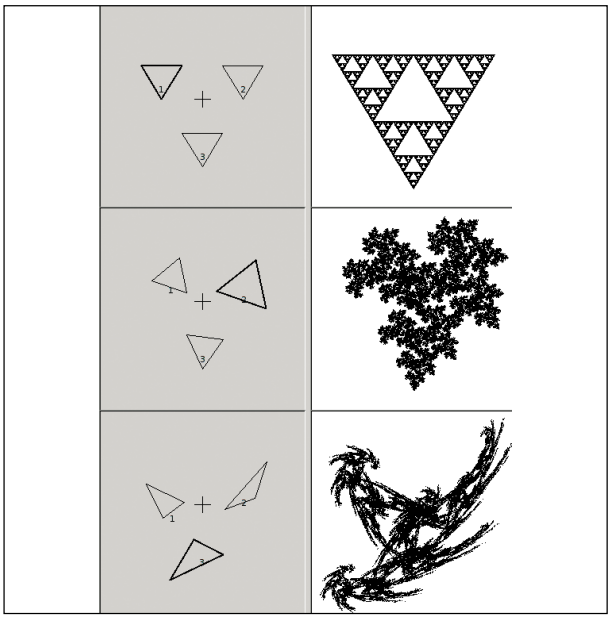
A ***GIMPressionist*** beépülő modul tulajdonképpen egy festői hatások létrehozására alkalmas rutingyűjtemény. A menüpont kiválasztása után lehetőségünk van különféle ecsetek, hordozófelületek és festési módok meghatározására, majd a ***GIMP*** a beállított értékek alapján megpróbálja a képet festménnyé alakítani. Az ecsetvonások iránya a képpontok különböző tulajdonságai alapján határozódik meg, és természetesen beállítható az is, hogy milyen hosszú vonásokkal dolgozzunk. Számos előre eltárolt beállítás közül válogathatunk kedvünkre, némelyikkel meglehetősen látványos képeket tudunk produkálni.

Az ***Oilify*** egyszerű hatású eszköz, aminek működése abban áll, hogy meghatározott méretű foltokkal helyettesíti a képpontok egy-egy csoportját. „Sajnos" az eredmény messze elmarad az olajfestmények által visszaadott színvilágtól és nem láthatunk a kinyomtatott képen festék által alkotott mintázatot sem. Néhány esetben (pl. képzőművészeti kiállítás reklámanyaga, absztrakt mintázatok létrehozása) azonban ez a hatás is megkönnyítheti munkánkat vagy segít kifejezni önmagunkat.

Lássuk, milyen lehetőségeket kínál még a program. A jobb egérgombos menüben láthatjuk, hogy korántsem értünk a lista végére. A ***Map*** almenüben látható hatások közös jellemzője, hogy a képeink átalakításához valamilyen más mintát vagy képet használnak fel.

A ***Fractal Trace...*** egy Mandelbrot-halmaz alapján alakítja át a képet. A ***fraktálok*** egyik jellegzetes tulajdonsága, hogy önmagukat tartalmazzák és rekurzív módon ha-

© Kiskapu Kft. Minden jog fenntartva



7. kép IFS fraktálok

tározzák meg. Tehát amikor egy képet egy fraktálra képezünk le, akkor a végeredmény az eredeti objektum ismétlése lesz egyre kisebb méretben és mindig más helyzetben.

A ***Bump Map...*** hatás alkalmazásakor a ***GIMP*** a kiválasztott képet (amely alapértelmezésben az éppen szerkesztett kép), mint felületi érdességet alkalmazza. Így akár készíthetünk vászon alátétet, vagy az eltolás (***X Offset*** és ***Y Offset***) változtatásával létrehozhatunk animációkat is.

A ***Displace*** hatással tengelyenként megadott képeken (az eltolás térképe) szereplő mértékben eltolhatjuk az adott kép egyes pontjait. Az eltolás során a térképként használt képpont világosságértékével (0.0 – 1.0) a ***GIMP*** megszorozza a megadott maximális eltolási értéket, majd az így kapottal megváltoztatja a kép megfelelő pontjának koordinátáit. Mindkét iránytengely mentén megadható más kép vagy a szerkesztetthez tartozó másik réteg. Az eltolási térképként használt rétegnek a képpel megegyező méretűnek kell lennie.

A ***Illusion...*** hatással különböző módon feloszthatjuk és az egyes darabokat meghatározott sorrendben, a szerkesztett kép középpontja körül elhelyezhetjük.

A ***Map Object...*** hatás segítségével a szerkesztett képet egy tárgy felületére feszíthetjük. A tárgy előre meghatározott forma lehet. A GIMP 2.0-ás változatában megtalálható a síklap, a gömb, a kocka és a henger. A tárgynak adhatunk anyagot, beállíthatunk fényforrást és a tárgy elhelyezkedését is megváltoztathatjuk. A kocka alakú tárgy esetében érdemes megemlíteni a külön megjelenő párbeszédfület, amelyen oldalanként meghatározhatjuk a kocka lapján megjelenítendő képet.

A ***Paper Tile...*** hasonló a korábban tárgyalt ***Glass Tile...*** hatáshoz, de itt nem üvegdarabokról van szó, hanem egyszerű síklapokról, melyek között a program hézagokat hagy. A végeredmény kis jóindulattal olyan, mintha a képet apró darabokból ragasztottuk volna össze.

A ***Small Tiles...*** a kép teljes területét felezéses módszerrel felosztja, majd az eredetit kicsinyítés után elhelyezi az ismétlődő területekre. A felosztás iránya és mértéke megváltoztatható a hatás beállítóablakában. Nos, egyelőre ennyit a különféle hatásokról, azonban érdemes megtekinteni a ***GIMP***-ben a honlapokhoz használható úgynevezett ***image-map*** előállító segédeszközt. Az ilyen térképek úgy képzelhetők el, hogy egy adott kép meghatározott területeihez más-más működést rendelünk. Előre meghatározzuk az érzékeny területeket, és ebben kap szerepet a ***GIMP***. A jobb egérgombos menüt megnyitva, a ***Web*** almenüben található meg ez a segédeszköz. Ahogyan az a 6-os képen is látható, meglehetősen összetett feladatokat oldhatunk meg vele. Képesek vagyunk kör vagy négyszög formájú területet kijelölni, vagy tetszőleges formát a sokszög elnevezésű eszközzel. Ez utóbbi jelenleg nem működik tökéletesen, mint ahogyan a varázspálca kiválasztása sem, mert működése után a modul teljesen használhatatlanná vált, nem frissítette a képet és a terület pontjait sem tárolta. Ettől eltekintve a kör és a négyszög jól használható, minekutána pedig a beépülő modul minden szükséges információt bekér, és a ***File -> Save as*** menüpontok választásával elmenthetjük az elkészített HTML forrást.

Képek a semmiből

A következő említésre méltó almenü a jobb egérgombos menüben a ***Render*** nevet viseli, itt aztán mindent megtalálhatunk, ami csak eszébe jutott már valakinek. Itt alapvetően olyan modulokat találunk, amelyek megadott paraméterek alapján állítanak elő valamilyen képet a „semmiből". Ebből az is következik, hogy használatukhoz nem igényelnek semmilyen más forrást.

Az elsőként említhetjük a két legegyszerűbb modult, a ***Plasma*** és a ***Solid noise*** megnevezésűt. A ***Plasma*** alapvetően szintén egy rekurzív területfelosztáson alapuló algoritmus. Egyszerű módon a terület négy csúcspontjában lévő, (kezdetben) véletlenszerűen meghatározott képpont átlagolásával kiszámítjuk a terület közepén elhelyezkedő pont színét. Ezt a színt egy kicsit megbolondítjuk valamilyen véletlen számmal, majd az aktuális területet felosztjuk négy egyenlő részre és a műveletet megismételjük. Mindez addig folytatódik, amíg a leképezendő terület oldalhosszúsága egész számmal kifejezhető. Ennek az algoritmusnak másik változata adja a ***Noise*** hatás alapját is.

A ***Flame*** és az ***IFSCompose*** fraktál alapú mintázatok előállítására használható. Érdemes kicsit közelebbről megismerkedni az ***IFS*** fraktálokkal, hiszen néhány háromszög tetszőleges elforgatásával, átméretezésével vagy áthelyezésével igen látványos mintákat állíthatunk elő. Néhány ilyen látható a 7-es képen.

Úgy gondolom, hogy erre a hónapra ennyi érdekesség és új ismeret bőven elegendő. Remélhetőleg mindenki használát látja ennek a leírásnak akár mindennapi munkája során, akár egyéb időtöltés keretében.

A következő számban megismerkedünk a még nem tárgyalt hatásokkal a ***Distorts*** és a ***Render*** menüpontokon keresztül, és megpróbálunk valamilyen összetett képet elkészíteni képzeletbeli reklámgrafikai stúdiónk számára. Addig is minden kedves olvasómnak kellemes alkotást kívánva búcsúzom.

Fábián Zoltán

© Kiskapu Kft. Minden jog fenntartva

A Perl és az adatbázisok (2. rész)

Szótár alapú adattárolás állomány-zárolással – az adatbáziskezelők fejlődésének egy elfelejtett lépcsőfoka.

A múlt hónapban egy közönséges szövegfájlban tároltunk egy egyszerű adatbázist. Bár az adattárolás ilyen megvalósítása ma sem számít túlhaladottnak, számos, a komoly rendszerek esetében egyáltalán nem elhanyagolható hiányossága felett átsiklottunk. Mentségünkre csak az szolgálhat, hogy a rendelkezésre álló eszközökkel nehezen hidalhattuk volna át a problémákat.
Ha adatainkat szövegként tároljuk, nincs külön adatbáziskezelő, amely azok épségéért, közvetlen módosításáért, illetve kezeléséért felelne. Mivel nem egy jól elhatárolt réteg végzi ezeket a feladatokat, hanem magának a programozónak kell minderről gondoskodnia, az ilyen rendszer teljesen rugalmatlan és nagyon nehezen bővíthető. A most bemutatásra kerülő módszernél adatbáziskezelő helyett, továbbra is egy függvénykönyvtárra fogunk támaszkodni, amivel azonban már elkerüljük az adatállomány közvetlen kezelését.
Ebben a hónapban a ***Berkeley*** adatbáziskezelőt fogjuk kipróbálni. Ez hash-táblával, vagy kiegyensúlyozott fával dolgozó, szöveges kulcs-érték párokon alapuló adattároló alrendszer. A Berkeley saját állomány-formátummal, a ***dbm***-mel dolgozik. Ezt a formátumot már nem módosíthatjuk közönséges szövegszerkesztővel, sőt a tartalmát sem jeleníthetjük meg ilyen módon.
A szöveges kulcs-érték párokból felépített szótár ötlete meglehetősen régi, és számos megvalósítása létezik, amelyek között szabad felhasználású és zárt forrású, egyaránt akad. Ezek rendszerint szerkezetileg is különböznek. Közülük a *Perl* az `ndbm`, `db`, `gdbm`, `sdbm` és `odbm` szabványokat támogatja. A modulként hozzáférhető megvalósítások abban valamennyien megegyeznek, hogy egy közönséges asszociatív tömbön végzett memória-műveleteket fordítanak le a háttérben alacsony szintű állománykezelési függvényekre. Ez nagyon kényelmes megoldás, hiszen egyetlen tömböt kell csupán kezelni, minden egyéb megy magától. Nem kell a karakterláncok hosszúságával törődni, és nincs határolójel, amely szerepelhetne bárhol. A tároló algoritmus hatékony, így egy rekord módosítása gyorsabb, mint a szöveges alapú adatbázisnál, nagy adatmennyiség esetén pedig a hash-tábla miatt a lekérdezés is sebesebb. Az egyetlen gyenge pont az új rekord felvétele lehet.
Perlben van egy függvény, amely egy változónevet köt össze egy osztállyal, melynek eredményeként a változón végzett bármilyen művelet a kérdéses osztály tagfüggvényeivel valósul meg. Így az adatbázison végzett munka három egyszerű lépésből áll. Hozzá kell kötni egy tömböt az adatállományt felügyelő osztályhoz, a tömbön el kell végezni a szükséges műveleteket, majd meg kell szüntetni a kötést, hogy minden adat a lemezre íródjon. Lássunk egy példát.

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

use strict;
use DB_File;

my %adatbazis;

# 1. lépés: adatfeltöltés
tie %adatbazis, 'DB_File', "elso.db", O_CREAT |
    ➥O_WRONLY or
    die "Nem tudom megnyitni az adatbázist.\n";

$adatbazis {'doggie'} = "kutyus";

untie %adatbazis;

# 2. lépés: lekérdezés
tie %adatbazis, 'DB_File', "elso.db", O_RDONLY
    ➥or
    die "Nem tudom megnyitni az adatbázist.\n";

print $adatbazis {'doggie'} . "\n";

untie %adatbazis;
```

Az első két sor – ahogy azt a sorozat korábbi tagjaiban is hangsúlyoztam – elengedhetetlen az átlátható és hibátlan ***Perl*** program írásához. A parancsértelmezőt a -w kapcsolóval hívjuk meg a figyelmeztető üzenetek megjelenítéséhez, majd használatba vesszük a `strict` modult. Az első újdonság a `DB_File` hívása. Ez egy olyan modul, amely tartalmaz egy osztályt a ***Berkeley*** típusú adatbázisok kezeléséhez.
A negyedik sorban a szigorú módnak megfelelően meghatározzuk a használni kívánt asszociatív tömb érvényességi

© Kiskapu Kft. Minden jog fenntartva

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

use strict;
use DB_File;
use Fcntl ':flock';

my %adatbazis;

# 1. lépés: adatfeltöltés
my $db = tie %adatbazis, 'DB_File',
"masodik.db", O_CREAT | O_RDWR or
    die "Nem tudom előkészíteni az
        ➥adatbázist.\n";
open ADATALLOMANY, "+<&=" . $db -> fd () or
    die "Nem tudom biztonságosan megnyitni az
        ➥állományt.\n";
flock ADATALLOMANY, LOCK_EX or
    die "Nem tudom megszerezni az egyedi
        ➥zárat.\n";

$adatbazis {'doggie'} = "kutya";

undef $db;
untie %adatbazis;
close ADATALLOMANY;

# 2. lépés: lekérdezés
tie %adatbazis, 'DB_File', "masodik.db",
O_RDONLY or
    die "Nem tudom előkészíteni az
        ➥adatbázist.\n";
open ADATALLOMANY, "<&=" . (tied %adatbazis) ->
    ➥fd () or
    die "Nem tudom biztonságosan megnyitni az
        ➥állományt.\n";
flock ADATALLOMANY, LOCK_SH or
    die "Nem tudom megszerezni a megosztott
        ➥zárat.\n";

print $adatbazis {'doggie'} . "\n";

untie %adatbazis;
close ADATALLOMANY;
```

körét – jelen esetben olyan, mintha globális változó volna. A program következő része két szakaszra bontható. Az elsőben megnyitja az ***elso.db*** adatállományt, és létrehoz benne egy értéket. A módszer „szótár jellegét" hangsúlyozandó egy angol szót használunk kulcsként, amelynek magyar megfelelője az érték. Az állomány lezárása után a második szakaszban ismét megnyitjuk azt, de már csak olvasásra, majd lekérdezzük az előbb felvett adatot, eleve feltételezve, hogy az létezik. Végül lezárjuk az adatbázist.

Az adatbázis megnyitása mindkét lépésben a `tie` függvénnyel történt, amely az első paraméterként átadott változót köti össze a másodikban meghatározott osztállyal. Mivel közben történik egy példányosítás, a függvény további paraméterei a konstruktornak adódnak át. Jelen esetben csak a megnyitási módot meghatározó állandókat adtuk meg (a továbbiakról a `DB_File` súgójában olvashatunk). A létrejött objektumot egyébként a `tie` vissza is adja, ám ezt itt nem használtuk fel. Végül mindkét esetben az `untie` segítségével szakítottuk meg a kapcsolatot a változó és az adatbázis között.

A módszer egyszerű, de ha két folyamat egyszerre fordul ugyanahhoz az állományhoz, versenyhelyzet alakulhat ki. Ilyenkor kiszámíthatatlan, hogy melyik utasításai jutnak érvényre. Ennek megakadályozására két megoldás létezik, az egyik az állományzárolás, a másik az `O_EXLOCK` jelző használata. Ez utóbbi sajnos nem minden rendszeren támogatott. Ilyenkor segít a UNIX `flock()` rendszerfüggvénye. Ez sajnos egy folyam-azonosítót vár paraméterként, amit nem kapunk meg a `tie` meghívásakor. Kapunk viszont egy objektumot, melynek egy tagfüggvénye visszaadja a megnyitott adatállomány leíróját. Ezt felhasználva az `open()` függvénnyel már készíthetünk egy folyam-azonosítót. Lássuk, hogyan néz ki az előző példa zárolással (*2. lista*).

Az `Fcntl` modul használata a `flock` függvény `LOCK_EX` és `LOCK_SH` állandói miatt szükséges. Az `open` második paramétere egy három részből álló karakterlánc. A „+<", vagy a „<" jelentése megnyitás olvasásra és írásra, illetve a csak olvasásra. Az „&=" azt jelzi, hogy nem a megnyitásra váró állomány neve következik, hanem egy állomány-leíró. Az első lépésben a `tie` által visszaadott objektum `fd()` elemfüggvénye szolgáltatja ezt az adatot. Mivel a második lépésben nem tároltuk külön változóban az objektumot, ezért ennek hivatkozását a tied függvény adja meg.

Az `open` függvényt közvetlenül a `flock()` követi. Ez két paramétert vár. Az első az említett folyam-azonosító, a második a zár típusát határozza meg. Egyedi zárat egyetlen folyamat szerezhet meg, így ezt íráskor szokás használni. A megosztott zárat párhuzamosan több folyamat is megkaphatja, ám ezalatt egyetlen másik sem szerezhet egyedi jogot a zárolásra. Mivel az olvasás egyszerre több folyamat számára is engedélyezhető, az írást viszont tiltani kell, ezt a módszert csak olvasó folyamatok szokták használni.

Van egy további csavar is a fenti példában. Bár előbb történik a kötés, csak utána a megnyitás, mégis előbb a kötést oldjuk fel, és utána zárjuk le a folyamot. Ha egy függvényt és az ellentettjét fordított sorrendben hívjuk meg, rendszerint hiba keletkezik. Esetünkben az `open`-nél nem történt valódi megnyitás, csupán egy folyam-azonosító készült egy meglévő állomány-leíróból, a `close`-nál viszont a lezárás valódi. Ezért ha hamarabb zárjuk le az állományt, mint ahogy a kötést megszüntetnénk, nem az asszociatív tömbben található adatok kerülnek az adatállományba.

A fenti megoldás majdnem tökéletes a versenyhelyzetek elkerülésére. Azért csak majdnem, mert a `tie` által meghívott konstruktor miután megnyitotta az adatállományt, még azelőtt végez egy kis lemezműveletet, hogy a vezérlés az `open` függvényre kerülne. Ezalatt a zárolás sajnos még nem él, de ezt ezzel a módszerrel nem is lehet elkerülni. Az egyedüli gyógyír a beépített zárolás, az `O_EXLOCK` lenne, viszont ez nem mindenhol elérhető.

Egy tábla rekordjai tetszőleges számú mezőből állhatnak, de egy adott táblában minden rekord ugyanannyi mezőt tartalmaz. Vajon lehet-e egy egész rekordot egyetlen kulcs-érték párba sűríteni? A kulcs kerülhet a tábla azonosító me-

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

use strict;
use DB_File;
use Fcntl ':flock';

die "Használat: " . $0 . " <adat fájl>\n" unless
    ➥1 == @ARGV;

my %adatbazis;
tie %adatbazis, 'DB_File', $ARGV[0], O_CREAT |
    ➥O_RDWR or
    die "Nem tudom előkészíteni az
        ➥adatbázist.\n";
open ADATALLOMANY, "+<&=" . (tied %adatbazis) ->
fd () or
    die "Nem tudom biztonságosan megnyitni az
        ➥állományt.\n";
flock ADATALLOMANY, LOCK_EX or
    die "Nem tudom megszerezni az egyedi
        ➥zárat.\n";

print "Neve                          : ";
my $nev = <STDIN>; chop $nev;
print "Telefonszám                   : ";
my $telszam = <STDIN>; chop $telszam;
print "Ezzel veheted le a lábáról : ";
my $szereti = <STDIN>; chop $szereti;

$adatbazis {$nev} = join (":", $telszam,
          ➥$szereti);

untie %adatbazis;
close ADATALLOMANY;
print "Az új lány (" . $nev . ") felvéve.\n";
```

3

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

use strict;
use DB_File;
use Fcntl ':flock';

die "Használat: " . $0 . " <adat fájl> <~lány
    ➥neve>\n" unless 2 == @ARGV;

my %adatbazis;
tie %adatbazis, 'DB_File', $ARGV[0], O_RDONLY or
    die "Nem tudom előkészíteni az
        ➥adatbázist.\n";
open ADATALLOMANY, "<&=" . (tied %adatbazis) ->
fd () or
    die "Nem tudom biztonságosan megnyitni az
        ➥állományt.\n";
flock ADATALLOMANY, LOCK_SH or
    die "Nem tudom megszerezni a megosztott
        ➥zárat.\n";

my ($kulcs, $ertek);
while (($kulcs, $ertek) = each %adatbazis) {
    if ($kulcs =~ /$ARGV[1]/) {
       print " Adatlap: " . $kulcs . "\n";
       print "==========" . "=" x length
             ➥($kulcs) . "\n";
       my ($telszam, $szereti) = split (/:/,
          ➥$ertek);
       print "  Telefonszám                  : "
             ➥. $telszam . "\n";
       print "  Ezzel veheted le a lábáról : "
             ➥. $szereti . "\n\n";
    }
}

untie %adatbazis;
close ADATALLOMANY;
```

4

© Kiskapu Kft. Minden jog fenntartva

zőjébe, és mivel a ***dbm*** nem szab határt az érték hosszára, az összes többi mezőt is be lehet ide rakni. Ezt pedig megoldható az egyszerű szöveges állománynál már látott elválasztó karakteres, vagy rögzített hosszúságú mezőkből álló láncokkal. Egy ilyen „öszvér" megoldás nemcsak gyorsabb, hanem rugalmasabb is mint egy sima szöveges adatbázis. Végezetül erre is lássunk egy példát. Két szkriptet fogunk elkészíteni. Az elsőnek (*3. lista*) az lesz a feladata, hogy az előző cikkben már látott adathalmazzal feltöltse adatbázisunkat, míg a második (*4. lista*) az adatok lekérdezésére szolgál. Az érték kettősponttal elválasztott mezőket fog tartalmazni, melyeket a `split()` / `join ()` párossal kezelünk. A fent látható programban nincsenek nagy meglepetések. Ez annak köszönhető, hogy a `tie` - `untie` páros segítségével az új elem felvétele mindössze egy változónak való értékadást jelent. Nem kell sokat morfondíroznunk ahhoz sem, hogy kitaláljuk, mi a megoldása a kettős kulcs problémának. Ez a gond ugyanis itt fel sem merül, hiszen ha az asszociatív tömb egy már létező elemére hivatkozunk, akkor az önműködően felülíródik. Az, hogy adott esetben valóban ez az elvárt működés, vagy mégis külön figyelmet kell fordítani a meglévő rekordokra, a feladattól függ.

A kiíratáskor a már megszokott kinézetet próbáltam követni. Természetesen itt nem adható meg, hogy a találat hányadik sorban történt, hiszen egy szótár mindig rendezetlen. Ennek megfelelően elképzelhető, hogy egy adatbázis megváltoztatása után a fent látható `each` szerkezet eltérő sorrendben adja vissza az adatokat. Ez azonban jelen esetben minket nem különösebben érint, legfeljebb két futtatás után más sorrendben látjuk viszont az adatokat.

Akinek van kedve, a fenti példákból kiindulva elkészítheti azt a szkriptet, amely képes egy Berkeley adatbázis egyik rekordját törölni. Mielőtt azonban nekilátnánk, érdemes egy pillantást vetni a `delete()` függvény leírására a ***perlfunc(1)*** kézikönyvlapon. Sok sikert a kísérletezéshez! A jövő hónapban már az SQL lesz terítéken.

Fülöp Balázs

© Kiskapu Kft. Minden jog fenntartva

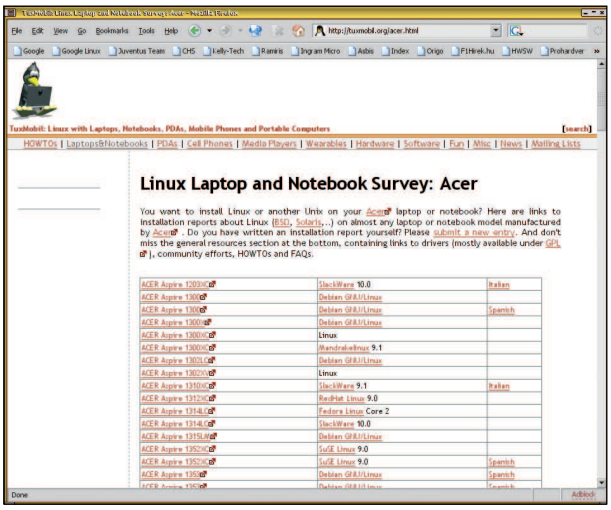
A pingvin igenis vándormadár

Avagy használjunk Linuxot mobil eszközökön – 1. rész

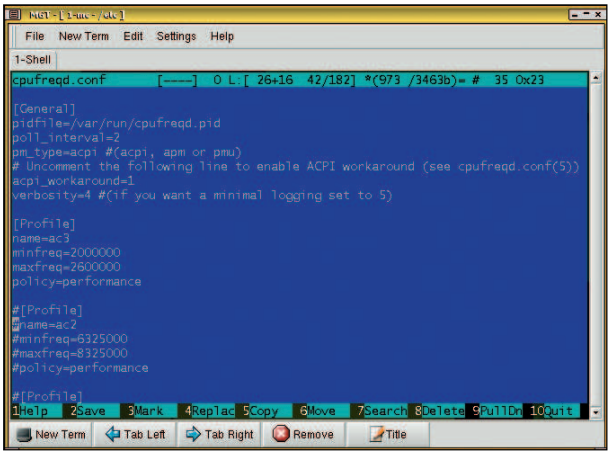
Napjainkban a különböző mobil eszközök fénykorukat élik. Ahova nézünk, mindenütt notebookok, tablet PC-k, PDA-k (digitális asszisztensek), mobiltelefonok, média lejátszók. Minden ilyen eszköz futtat valamilyen operációs rendszert. Némelyiket teljesen hagyományos, az asztali gépeknél is alkalmazott operációs rendszerrel látják el, mások annak egy módosított változatát futtatják, de akad olyan is, amit teljesen speciális, egyedileg hozzá tervezett rendszerrel szállítanak. Egy megszállott Linux felhasználóban persze rögtön felmerül a kérdés, nem lehetne-e az ilyen eszközökön is Linuxot használni? A válasz – nem túl meglepő módon – de lehet. Sőt, vannak olyan mobil eszközök, amelyek már gyárilag valamilyen Linux rendszert futtatnak, bizonyítva ezzel, hogy a Linux komoly alternatíva ezen a területen is.
Cikksorozatomban egyrészt áttekintést kívánok adni arról, hogy milyen mobil eszközökre milyen linuxos megoldás létezik, melyiknél milyen szolgáltatásokra és buktatókra számíthatunk. A sorozatban bemutatom, hogy jómagam miféle tapasztalatokat szereztem Linuxot használva a saját notebookommal. Megpróbálok arra törekedni, hogy az anyag ne csak a hozzáértők számára legyen hasznos, hanem általa a kezdő felhasználók is kedvet kapjanak ahhoz, hogy kipróbálják, milyen is Linuxot használni az előbb felsorolt eszközök valamelyikén.

Linuxot a notebookomra!

Kezdjük talán a legkézenfekvőbb mobil eszközzel, amelyre Linuxot telepíthetünk, vagyis egy notebookkal. A telepítés folyamata ugyan nem sokban különbözik a szokásostól, de az a kevés különbség alaposan megnehezítheti az ember életét. Én jelenleg Debiant használok a saját hordozható gépemen, így értelemszerűen ezt fogom bemutatni, de a többi Linux kiadás is teljesen hasonló módon telepíthető, használható.
A csomaggyűjtemények összetétele a különböző terjesztésekben meglehetősen hasonló. Ha egy-egy programot nem találnánk kedvenc terjesztésünkben, a jól ismert ➲ ***sourceforge.net***, vagy ➲ ***freshmeat.net*** biztosan segítségünkre lesz.
Aki egy konkrét géptípussal kapcsolatban kíváncsi mások tapasztalataira, annak érdemes körülnéznie a ➲ ***tuxmobile.org*** oldalon, ahol gyártó és típus szerinti bontásban rengeteg mobil eszközzel kapcsolatos tapasztalat van összegyűjtve. Itt egészen biztosan megtaláljuk, hogy miért nem működik az a fránya infra-csatoló, vagy miért nem lehet a beépített



modemmel tárcsázni. Mielőtt tehát nekivágunk a telepítésnek, érdemes átfutni ezt az oldalt, mert sok bosszúságtól kímélhetjük meg magunkat.
Akkor tehát kezdjük az elején. Mint említettem, Debiant fogunk telepíteni. Először töltsük le a legfrissebb telepítő CD-t a ➲ ***debian.org*** webhelyről, írjuk fel egy lemezre és bootoljunk be róla. Én a legfrissebb ***Woody*** CD-t töltöttem le, de lehet próbálkozni a ***Sarge***-al is, amit jelenleg még tesztelnek. (A ***Sarge*** lesz a következő Debian verzió.)
Rendszerbetöltés után a telepítést a `bf24` opcióval indítsuk. Ebben az esetben a ***Woody*** a 2.4.18-as kernellel települ, így egy viszonylag korszerűnek mondható rendszermagot használhatunk már a telepítés során is. Sajnos van olyan gép – és itt nem csak notebookokra kell gondolni –, amelyik valamiért nem szereti a 2.4-es rendszermagot, amit nemes egyszerűséggel úgy juttat kifejezésre, hogy telepítéskor lefagy. Ha ilyet tapasztalnánk, akkor – minden szomorúságunk ellenére – ne adjuk fel a küzdelmet. Telepítsünk a 2.2-es kernellel, később úgyis kicseréljük.
A telepítés ezután a szokott módon zajlik. Aki telepített már Debiant, tudja miről beszélek, aki még nem, az meg úgyis megtudja... A telepítés talán legfontosabb mozzanata, hogy hálózati kártyánkat életre keltsük és internet csatlakozásunkat megfelelően beállítsuk, mert erre a későbbiekben nagy szükségünk lesz a telepítés és a rendszer használata folyamán. Ma már a notebookok 99%-át valamilyen beépített hálózati csatla-



kozóval szállítják, amely az esetek túlnyomó többségében a Realtek 8139-es valamilyen változata, vagy Intel PRO/100. Előfordulnak 3com lapkával szerelt gépek is.
A 2.4-es rendszermag mindegyiket támogatja, így a hálózati csatlakozással nagy valószínűséggel nem lesz probléma. Ha a gépben nincs beépített Ethernet csatoló, vagy nem ismerte fel azt a telepítő, használhatunk PCMCIA kártyát is.
És ezzel el is érkeztünk az első lényeges ponthoz. Aki eddig csal asztali gépekre telepített Debiant, az ösztönösen igennel válaszol a „Kívánja, hogy a telepítés végeztével a pcmcia csomagok eltávolításra kerüljenek?” kérdésre. Ez ilyenkor nem jó ötlet. Szinte biztos, hogy egy notebook esetében még szükségünk lesz erre a lehetőségre.
Ha sikeresen túlestünk a hálózati kártya beállításán, akkor jó esély van rá, hogy a továbbiakban értelmes kérdésekre adott értelmes válaszokkal viszonylag gyorsan végzünk a telepítéssel.
Ha nem sikerült a hálózati kártyát felismertetni a rendszerrel, vagy valamilyen vezeték nélküli megoldást kívánunk használni, akkor sincs semmi katasztrófa. Telepítsük a rendszert hálózati támogatás nélkül, a hálózatot pedig állítsuk be később. Vezeték nélküli hálózat esetén ez eleve csak így oldható meg, mivel a hálózati adapter használatához telepítenünk kell néhány egyéb kiegészítő csomagot. Ezekről a későbbiekben még ejtünk szót.
Ha feltelepült a rendszer és „alapjáraton” működőképes, akkor tegyünk fel még néhány hasznos csomagot, és végezzünk el néhány kézenfekvő módosítást, mielőtt továbbmennénk. Először is érdemes egy Midnight Commandert telepíteni. Ezt pillanatok alatt megtehetjük az `apt-get install mc` paranccsal. (A csomag a telepítő CD-n megtalálható.) Az `mc` segítségével váltsunk a */etc/apt* könyvtárba és nyissuk meg az ott található ***sources.list*** állományt. Debian alatt ez az állomány határozza meg, hogy a csomagokat a telepítés során milyen forrásból szerezze be a rendszer. Mivel ma már nagyon sok helyen hozzáférhető szélessávú Internet elérés, a továbbiakban feltételezem, hogy az Olvasó is ilyent használ. Ha mégsem, a ***woody*** terjesztés helyett talán hasznosabb a a ***sarge***-ot letölteni, mert az lényegesen frissebb csomaggyűjteményt tartalmaz.
Visszatérve a ***sources.list*** állományra, nyissuk meg szerkesztésre és adjunk hozzá pár új forrást, az eddigieken kívül. Jómagam a sid csomaggyűjteményt használom, amely a fejlesztésből közvetlenül kikerülő csomagokat tartalmazza.

Néhány szó az aptitude használatáról

A program indítása nem meglepő módon az `aptitude` paranccsal történik. Ha elindult, akkor a csomagokat állapotuk és rendeltetésük szerint csoportosítva láthatjuk a nyitó képernyőn. A / karakterrel kereshetünk (a magyar billentyűzetkiosztás szerint SHIFT+6), a keresett szóra ismételt keresést pedig a \ (ALT GR+Q) karakterrel kezdeményezhetünk. Egy adott csomag leírását és függőségeit a csomagra lépve és enter-t ütve kapjuk, míg innen a Q gombbal térhetünk vissza. Ha egy csomagot telepíteni szeretnénk, akkor azt a + karakterrel tehetjük meg, törlés esetén pedig a - karaktert kell használni. Ha egy csomagot telepítésre, vagy törlésre jelöltünk ki, de közben meggondoltuk magunkat, úgy az = karakterrel a kiindulási állapothoz térhetünk vissza.
Az U billentyű lenyomása esetén a program frissíti a csomaglistát, amit a ***/etc/apt/sources.list*** állományban beállítottunk. Ha végeztünk a csomagok összeállításával, akkor a G billentyű lenyomására a telepítő összeállítja a változó csomagok listáját. Zölddel jelöli a telepítendő, lilával a törlendő, kékkel a frissítendő, pirossal a törött csomagokat. Amennyiben az összes törött csomagot kijavítottuk – ha volt egyáltalán ilyen –, akkor utána ismételten megnyomva a G billentyűt indul a letöltés és telepítés.
Ha végeztünk, a programból a Q billentyűvel léphetünk ki.

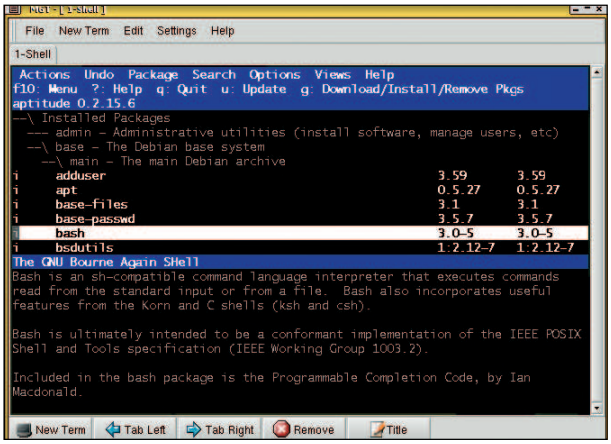
Ez egyfelől hasznos, mert nagyon sűrűn érkeznek a frissítések, másrészt előfordulhatnak hibás csomagok. Aki ezt a kockázatot csökkenteni szeretné, annak megint azt tudom javasolni, hogy használja a ***Sarge*** gyűjteményt.
Aki a sid mellett döntött, adja a következő sorokat a ***sources.list*** állományhoz:

```
deb http://ftp.hu.debian.org/debian sid main non-
➥free contrib
deb http://non-us.debian.org/debian-non-US
➥sid/non-US main non-free contrib
```

A biztonságosabb ***Sarge***-ot választók a fenti sorokban a sid szót értelemszerűen cseréljék ***Sarge***-ra.
A csomaglista kibővítése után rögtön töltsük is le az új csomagokat az `apt-get update` paranccsal. Hogy a továbbiakban megkönnyítsük a csomagok telepítését és karbantartását, javaslom, hogy telepítsük fel az `aptitude` nevű, szöveges felületű csomagkarbantartó programot, amely nagyon ügyesen kezeli a csomagok telepítésének folyamatát. Jól lehet benne keresni, sok információt ad a csomagokról, jól kezeli a frissítéseket, egyszóval megkönnyíti az életünket.
Ha idáig eljutottunk és működik minden eszközünk, ami a telepítéshez szükséges, akkor már nagyon jól állunk. Gyakorlatilag van egy teljesen jól használható rendszerünk és kezdhetjük az érdemi munkát.
Következő lépésként töltsünk le egy friss rendszermagot. Én a 2.6-os sorozat legfrissebb változatát javaslom, mert rengeteg olyan eszközhöz nyújt támogatást, amelyekhez a régebbi 2.4-es verzió nem, vagy csak nagy küzdelmek

© Kiskapu Kft. Minden jog fenntartva

© Kiskapu Kft. Minden jog fenntartva



árán. A rendszermagot kétféleképpen telepíthetjük. Vagy letöltünk egy ún. kernel image-et és telepítjük, vagy letöltünk egy kernel forrást és magunk fordítjuk le.
Utóbbi talán a kezdő felhasználóknak túlzottan nagy falat lehet, de aki egy kicsit utánaolvas a dolognak, az hamar rájön, hogy ez nem is olyan nagy ördöngösség. Emlékeim szerint nem is kell messzire mennie annak, aki jó leírást szeretne a témában találni. Lapozza fel a Linuxvilág magazin korábbi számait, itt biztos megtalál mindent.
Én saját fordítású kernelt használok, azt is úgy, hogy a szükséges szolgáltatásokat belefordítom, és nem modulként használom őket. Úgy gondolom, egy egyedi konfiguráció esetén az kevesebb gonddal jár, mivel nem kell állandóan azzal foglalkoznom, hogy ellenőrizzem, a megfelelő modul rendelkezésre áll-e. Ha pedig változik a gépem kiépítése, hát annyi baj legyen, fordítok hozzá egy új rendszermagot.

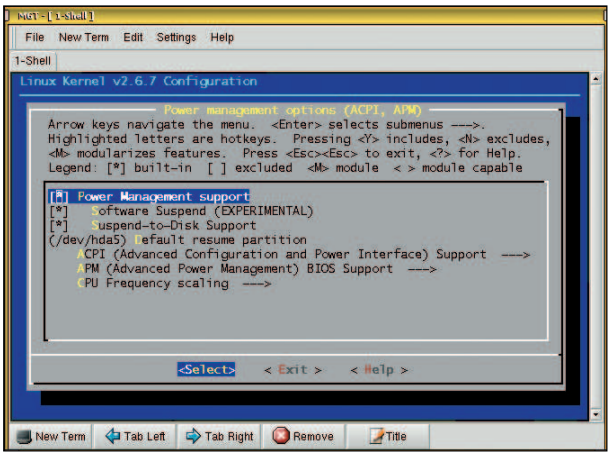
Energiagazdálkodás

Első konkrét témánk legyen az energiagazdálkodás, hiszen egy notebooknál ez kiemelkedő jelentőséggel bír. A 2.6-os rendszermag több energiatakarékossági funkciót is rendelkezésünkre bocsát. Egyfelől támogatja az ACPI energiagazdálkodási állapotokat (ilyen például a standby mód, a hibernálás, vagy a suspend), másfelől a normál asztali processzorral szerelt gépeknél is lehetővé teszi a processzor frekvenciájának dinamikus beállítását a gép terheltségétől és az akkumulátor töltöttségi szintjétől függően.
Az energiagazdálkodási állapotok támogatását a rendszermagban a Power management options (ACPI, APM) menüpont alatt tudjuk bekapcsolni. Akár a többit, ezeket a szolgáltatásokat is érdemes rendszermagba belefordítani, hogy mindig rendelkezésre álljanak.
A Software Suspend opció beállításával lehetőségünk nyílik arra, hogy munka közben bármikor leállítsuk a gépet úgy, hogy a következő bekapcsoláskor a megfelelő paraméterekkel indítva a rendszert pontosan ugyanabba az állapotba kerüljünk vissza, amelyben leállítottuk a gépet. Ilyenkor a rendszer az aktív csereterületen (swap partíción) készít egy állapotmentést amelynek helyét a következő indításkor megadva a rendszermag visszatölti a mentett állapotot.
A Software suspend szolgáltatás kihasználásához be kell fordítani a kernelbe a Suspend-to-Disk support opciót is, amely a Software suspend pont alatt található.

Ugyanitt a Default resume partition értéknek adjuk meg az aktív swap partíciónkat (pl. `/dev/hda2`), amelynek hollétét a legegyszerűbben az */etc/fstab* állományból tudjuk kideríteni.
A rendszerre vonatkozó ACPI beállításokat a ACPI (Advanced Configuration and Power Interface) Support menüpont alatt tudjuk elvégezni. Itt állíthatjuk be, hogy a Linuxunk támogassa az ACPI energiagazdálkodási állapotait (Sleep states), illetve a rendszer hálózati kapcsolatának, akkumulátorának, hőviszonyainak állapotát lekérdező eszközöket. Érdemes ezeket bekapcsolni, nélkülük ugyanis az olyan programok mint a telep töltöttség visszajelző, hőmérő, vagy a ventilátorokat ellenőrző rendszerek nem fogják megtalálni a megfelelő adatforrást.
A rendszerre vonatkozó APM beállításokat az APM Bios Support menüpont alatt adhatjuk meg. Amennyiben rendszerünk ACPI támogatással rendelkezik – ez ma már minden új gépre igaz – akkor ezekre a beállításokra nem feltétlenül lesz szükségünk. Megfigyeléseim szerint néha előfordul, hogy a leállítás utáni önműködő kikapcsoláshoz szükséges lehet a „Use real mode APM BIOS call to power off” modul rendszermagba való befordítása is.
A következő bekezdést szenteljük a 2.6-os rendszermag egyik újdonságának, a dinamikusan állítható processzorsebességnek. Az ezzel kapcsolatos lehetőségeket a CPU Frequency scaling menüpont alatt találjuk. Fordításkor jelöljük meg ezt a modult, illetve a 'powersave' governor Support, */proc/cpufreq* interface, 'userspace' governor for userspace frequency scaling, CPU frequency table helpers és ACPI Processor P-States driver modulokat. Az utóbbi esetében meg kell adnunk a saját gépünkben található processzorhoz és annak szolgáltatásaihoz tartozó (általában AMD, vagy Intel) modulokat is. Ha mindezt megtettük, akkor a későbbiekben telepítésre kerülő felhasználói programokkal vezérelhetjük majd a gép processzorának frekvenciáját. Mindez ezért hasznos, mert ha mobil gépünket asztali processzorral szerelték, amely alapból nem támogatja a dinamikus frekvenciaállítást, akkor is képesek leszünk – igaz csak alapszinten – változtatni a központi feldolgozóegység sebességét. Ezzel pedig jelentősen növelhető az akkumulátor üzemideje.

Az energiakezelés támogatása a felhasználói programok oldaláról

Mostanra tehát a rendszermagot felkészítettük arra, hogy az energiagazdálkodással kapcsolatos különböző kiegészítőket támogassa. Most vessünk egy pillantást arra, miként is lehet ezeket a szolgáltatásokat kihasználni.
Kezdjük talán az ACPI energiagazdálkodási állapotaival. Három fő energiatakarékos állapot létezik az ACPI szabvány szerint, sorjában az S1, az S3, és az S4. Az S1 állapot a Standby, vagy Power-On Suspend nevet viseli. Ebben az állapotban a gép bekapcsolt állapotban marad, a processzor dolgozik, a memória él, de a használaton kívüli eszközök kikapcsolnak. Lekapcsol a monitor háttérvilágítása, leállnak a nem használt kártyák (modemek, hálózati kártyák). Ebben az állapotban az akkumulátor továbbra is teljesítményt ad le, tehát hálózati tápfeszültség nélkül a gép – ugyan jelentősen hosszabb idő után – de mindenképpen lemeríti azt.



Az S3 állapot az ún. Suspend to RAM mód. Ilyenkor a gép a saját állapotáról mentett adatokat a memóriában helyezi el. Ez az S1-nél „mélyebb nyugalmi állapotot” jelent, mivel ebben esetben valamennyi hardvereszköz a legalacsonyabb energiafogyasztású állapotába kerül. Kivétel természetesen a memória, amelyhez ebben az esetben is biztosítani kell a frissítést. Ezzel az S1 állapothoz képest sokkal nagyobb energiamegtakarítást érhetünk el, de a gépnek még mindig van egy minimális energiaszükséglete.
Az S4 állapot, amelyet Suspend to Disk állapotnak nevezünk, lehetővé teszi, hogy teljesen leállítsuk a gépet, előtte lemezre mentve a rendszer állapotát. Az S4 állapot elérésekor egy képállomány készül a memóriáról és a rendszerállapotról, amely a kijelölt swap partícióra íródik. A gép ezután leáll. Amikor újra bekapcsoljuk, akkor a képállomány helyét (vagyis a kérdéses csereterületet) megadva, a rendszer beolvassa a korábban mentett állományt és visszaállítja az eredeti helyzetet.
A fent leírt energiagazdálkodási állapotok elérhetők különböző segédprogramokon keresztül is, de kiválthatók egyszerűen az `echo állapotszám > /proc/acpi/sleep` parancs futtatásával is. Ilyenkor a rendszer a megadott sorszámú (S1, S3 vagy S4) állapotba kerül. S4, vagyis Suspend to Disk esetén újraindításkor paraméterként át kell adnunk a rendszermagnak a csereterület helyét a következ módon:

```
resume=/dev/swap_particio_helye
```

A fejlesztők felhívják a figyelmet arra, hogy az említett magfunkciók jelenleg még fejlesztés alatt állnak, így sajnos előfordulhatnak akár adatvesztéssel is járó hibák. Az is lényeges hangsúlyozni, hogy S4-es állapotból való ébredés során adatvesztés léphet fel, ha a hardver összeállítása közben megváltozott, vagy a lemez tartalma módosult.

Dinamikus órajelvezérlés

A dinamikus processzorvezérléshez két csomag telepítésére lesz szükségünk. Az egyik a `cpufreqd`, a másik a `cpudyn`. Előbbi felelős a processzor kihasználtságának, az akkumulátor töltöttségének és a tápfeszültség állapotának figyeléséért, míg utóbbi a processzor és lemezek vezérléséért felel.
A `cpufreqd` csomaghoz tartozik egy beállító állomány (*/etc/cpufreqd.conf*), amelyben megadhatjuk az egyes terhelési és töltöttségi szintekhez tartozó sebességeket. Először létre kell hoznunk szabályokat (Rule kulcsszó). Minden szabálynak egyedi neve kell, hogy legyen, meg kell adni a hozzá tartozó tápfeszültség-állapotot (van külső táplálás vagy nincs), az akkumulátor töltöttségének mértékét (ez egy intervallum), és a processzorkihasználtság nagyságát. Megadható még a futó programok neve, valamint a használt energiagazdálkodási profil.

```
[Rule]
name=pelda_szabaly
ac=off
battery_interval=50-70
cpu_interval=30-60
programs=xine,mplayer
profile=pelda_profil
```

Mint látható, a fenti szabály a `pelda_szabaly` nevet kapta. Akkor lép életbe, ha nincs a gép külső feszültségforrásra kapcsolva, az akkumulátorok 50-70 százalékig töltöttek, a processzor kihasználtsága 30-60 százalékos és a `xine`, vagy az `mplayer` médialejátszó fut. Ezeket a feltételeket a rendszer a következő módon súlyozza. A tápellátás nyolcszoros, a processzor terheltsége négyszeres, az akku állapota kétszeres, míg a futó programok egyszeres szorzóval számítanak. Azt a szabályt fogja a rendszer használni az adott pillanatban, amelyik a legjobban illik az állapotra, azaz, amelyik adott pillanatban a legtöbb pontot kapta. Amennyiben azonos pontszámú szabályokat találna, úgy a listában előbb szereplő kerül alkalmazásra.
A profil megadása során négy értéket kell beállítanunk. A profil nevét, a legnagyobb frekvenciát, a legkisebb frekvenciát és a vonatkozó házirendet. Utóbbi lehet `performance`, vagy `powersave`, tehát vagy a teljesítményt tekintjük elsődleges szempontnak, vagy az energiatakarékosságot.

```
[Profile]
name=pelda_profil
minfreq=1325000
maxfreq=2600000
policy=performance
```

A fenti példa a `pelda_profil` nevet kapta, a minimális processzorsebesség 1,325 GHz, a maximális 2,6 GHz és az elsődleges szempont pedig a teljesítményigény kiszolgálása.
Amennyiben elkészültünk a módosításokkal, úgy a */etc/init.d/cpufreqd* és */etc/init.d/cpudyn* szkriptek újraindítása után a rendszer a már módosított beállításokat fogja használni.
Kezdetnek tehát telepítettünk egy új Debian rendszert, és beállítottuk az energiagazdálkodását. A cikksorozat következő részeiben további érdekes és hasznos beállításokkal fogunk megismerkedni. Addig is mindenki használja az új rendszerét, próbálgassa a különböző beállításokat, esetleg a cikkben említett weboldalak alapján próbálja önállóan működésre bírni a hardvereszközöket.

Illés Viktor

© Kiskapu Kft. Minden jog fenntartva

© Kiskapu Kft. Minden jog fenntartva

Az akkumulátoros üzemidő növelése a Laptop Mode segítségével

A merevlemez gazdaságos használatával meghosszabbíthatjuk hordozható gépünk akkumulátoros üzemidejét.

A hordozható gépek megadják nekünk azt a szabadságot, hogy bárhol, bármilyen feladatunkat elvégezhetjük. Sajnos, ha akkumulátoruk lemerül, az élvezetnek vége. Szerencsére több lehetőségünk van az energiával való takarékoskodásra és az akkumulátoros üzemidő növelésére. Megtehetjük például, hogy csökkentjük a processzor sebességét, tompítjuk a kijelző háttérvilágítását vagy lekapcsoljuk a merevlemezt. Az első két dolog eddig is remekül működött ***Linux*** alatt is, ám a merevlemez lekapcsolása csak nehézkesen ment. Még ha sikerült is megoldani a leállítást, ez az állapot sosem tartott elég sokáig ahhoz, hogy érdemleges energia-megtakarítást eredményezzen. Írásomban a ***Laptop Mode***-ot szeretném ismertetni, a ***Linux*** rendszermag egy nemrég megjelent elemét, amely jól használható megoldást biztosít a merevlemez leállítására. Itt csak a 2.6-os ***Linux*** rendszermaghoz készült változatról lesz szó. A ***Laptop Mode*** a 2.4-es változathoz is létezik, de némileg eltérő kiadásban.

A merevlemezek és az akkumulátoros üzemidő kapcsolata

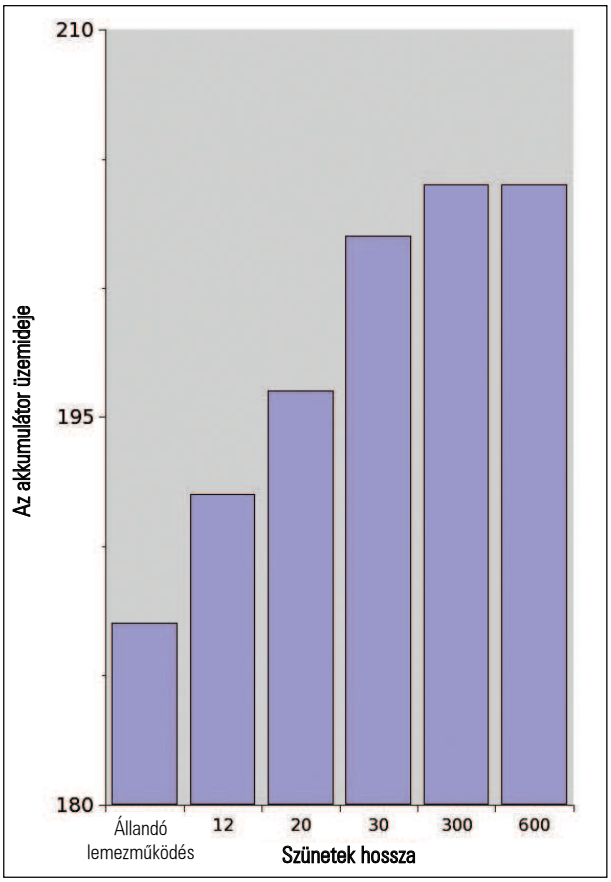
Vajon üzemidő tekintetében mekkora nyereséggel jár a merevlemez lekapcsolása? Próbáljuk meg kiszámítani! Egy átlagos mai hordozható gép lítium-ionos akkumulátora 50–100 wattóra energiát képes tárolni, ami 2-4 óra üzemidőhöz elegendő. Tegyük fel, a saját gépünk akkumulátora 50 wattórás. Ha az akkumulátor a merevlemez bekapcsolt állapota mellett 3,5 órás üzemidőt képes biztosítani, akkor az átlagos energiahasználat 50/3,5 = 14,3 watt. Tegyük fel, hogy a gépben szintén átlagos szintű a merevlemez használata, amely tétlen állapotban 0,9, készenléti módban pedig 0,3 wattot fogyaszt. Az energiahasználatot tehát elméletileg 0,6 watt megtakarításával 13,7 wattra tudjuk mérsékelni. Ezzel az akkumulátoros üzemidő 50/13,7 = 3 órára és 39 percre nő. A nyereség mindig a megtakarított energia és a teljes energiafelvétel viszonyától függ. Esetünkben a leállítással a teljes fogyasztásnak hozzávetőlegesen 4 %-át tudjuk megspórolni, vagyis az üzemidőben is hasonló mértékű növekedésre számíthatunk.

Ennyit az elméletről, nézzünk inkább néhány valódi adatot. Kölcsönkértem egyik barátom ***Apple PowerBook G4*** gépét, feltettem rá a ***Debian GNU/Linuxot*** 2.6.6-os rendszermaggal, és végeztem néhány kísérletet. Arra voltam kíváncsi, hogy legfeljebb mekkora nyereséget lehet elérni az akkumulátoros üzemidőben, és ehhez mekkora időtartamokra kell lekapcsolni a lemezt. Viszonylag jó eredményt vártam, hiszen a gépben egy nagy fogyasztású, 5400-as fordulaton üzemelő lemez volt, és a rendszerből kiszedtem az ***X*** kiszolgálót és az összes démont. Írtam egy teljesítménymérő programot, amely minden órában azonos mennyiségű lemezes be/kiviteli műveletet hajt végre, de úgy, hogy meg lehet adni a be/kiviteli műveletlöketek közötti szünetek hosszát. A tétlen időszakokban a mérőprogram leállítja a merevlemezt. Többféle tétlenségi időhosszal is futtattam a programot, miközben az ***APM*** által szolgáltatott adatok alapján becsültem meg a várható üzemidőt.

A kísérletet folyamatosan működő lemezzel, illetve a löketek közötti idő hosszát 12 másodperc és 10 perc közötti értékekre állítva végeztem el. Eredményeim az *1. ábrán* láthatók. Mint az ábráról is kitűnik, ha 30 másodpercenként átlagosan egynél kevesebb be/kiviteli műveletet végez a gép, már számottevő megtakarítást érhetünk el. Meglepő, hiszen a lemez felpörgetése sok energiát igényel, nem igaz? Nem, valójában nem így van. Ha a lemez két másodperc alatt felpörög, azzal nagyjából annyi energiát vesz fel, amennyit nyolc másodpercig tartó tétlen állapotában használna fel. Ha tehát legalább kilenc másodpercig leállítva tudjuk tartani a lemezt, már nyertünk valamennyit. A löketek közötti 30 másodperces időköz viszonylag hosszú leállást jelent, ezért kaptam ilyen kedvező eredményt.

A Laptop Mode

A ***Laptop Mode*** lényegében a ***Linux*** rendszermag egy beállítása, amely a lemezes be/kiviteli műveletek időbeli elosztását változtatja meg. A ***Linux*** normál esetben kisebb, időben jól elosztott adagokban végzi el a lemezes be/kiviteleket. Ha viszont a lemez, ámbár kényelmesen is, de folyamatosan dolgozik, akkor soha nem tud leállni, és értékes energiát veszítünk. A hordozható gépeknél a lemezműveleteket rövid időszeleteken belül kell végrehajtani, majd ezek között hosszabb szünetet kell tartani, ahogy az a kísérletemnél is történt. Ha engedélyezzük a ***Laptop Mode-ot***, a ***Linux***



1. ábra Az akkumulátoros üzemidőt vizsgáló kísérlet eredménye

is pontosan ezt teszi. A lemezműveletek között ez esetben akár 10 perc is eltelhet, így komoly növekedést érhetünk el az akkumulátoros üzemidőt tekintve.

Hogyan működik?

Tekintsük át azt, hogy a ***Laptop Mode*** hogyan éri el ezt a szokásostól eltérő be/kiviteli viselkedést. Ha tétlen időszakokat akarunk létrehozni, akkor az aktív időszakaszokban a lehető legtöbb be/kiviteli műveletet végre kell hajtanunk. Ha azt aktív időszak véget ért, akkor a lehető leghosszabb ideig vissza kell tartanunk az írási és az olvasási kérelmeket. Az aktív időszakokban meg kell próbálnunk némi pluszmunkát végezni. Először is, bizonyos mértékű előreolvasást kell végrehajtanunk. Ha az adatokra a tétlen időszakban valóban szükség lesz, akkor megtakarítottunk egy felpörgetést. A ***Laptop Mode*** alapesetben 4 MB-ot olvas előre. Ugyancsak fontos, hogy az aktív időszak végén minden változást írjunk is ki a lemezre. Ezzel biztosítani tudjuk az adatok biztonságát. Ha leáll a lemez, tudjuk, hogy a leállásig elvégzett munkánk már biztonságban van.

A tétlen időszakokban az írás az egyetlen visszatartható művelettípus, feltéve, hogy a kiírandó adatokat tudjuk tárolni a memóriában – ezt is csak addig tehetjük, amíg a memória meg nem telik. Sajnos ezt elmondani könnyebb, mint megvalósítani, a ***Linux*** ugyanis számos különböző helyről indít írásra vonatkozó kéréseket, ezeket aztán mind úgy kell módosítani, hogy lehetővé tegyék az írások visszatartását.

Az első és legfontosabb a módosított, vagyis piszkos adatok kezelése. Normál esetben, ha egy gyorstárazott lemezlap több mint 30 másodperce módosult, akkor elavultnak számít, és a pdflush démon kiírja a lemezre. Szerencsére az elavulási időtartam hossza módosítható (***/proc/sys/vm/dirty_expire_centisecs***). A ***Laptop Mode*** itt tíz perces időtartamot ad meg, vagyis a módosítások akár ennyi ideig is a memóriában maradhatnak, mielőtt a lemezre kerülnének. Mivel minden aktív időszak szinkronizálással végződik, a tétlen időszakok piszkos lapok létezése nélkül kezdődnek. A tétlen időszakok első tíz percében tehát biztosak lehetünk abban, hogy elavulás miatt egyetlen lap sem íródik vissza a lemezre.

A második fontos módosítás a naplózó fájlrendszerekre vonatkozik, hiszen ezek önmagukban is rengeteg lemezműveletet végeznek. A ***Laptop Mode*** által támogatott naplózó fájlrendszerek nagy részénél a fájlrendszerre vonatkozó változások egy öt másodpercen belüli írási műveletet váltanak ki. Az ***ext3***-as fájlrendszernél például a fájlrendszerre vonatkozó tranzakcióknak van egy maximális élettartamuk, ennek lejártával azonnal a lemezre kerülnek, ami értelemszerűen egy írási műveletet jelent. A maximális élettartam a `mount commit` kapcsolójával adható meg. Ha leválasztunk és újra befűzünk egy fájlrendszert úgy, hogy ezt az élettartamot tíz percre állítjuk, akkor az ***ext3*** nem fog tranzakciókat kiírni a tétlen időszakok alatt. Ismétlem, a tétlen időszakokat szinkronizálás előzi meg, vagyis a tétlen időszakok kezdetekor nincsenek nyitva hagyott tranzakciók. A ***Laptop Mode*** hasonló módosításokkal él a ***ReiserFS*** és az ***XFS*** esetében is.

Az utolsó változás a Linux memóriakezelését érinti. Ha tétlen időszakban nagymennyiségű memória lefoglalására kerül sor, a memóriakezelőnek ki kell választania néhány eldobandó lapot. Előfordulhat, hogy olyan lapot választ, amelynek tartalmát eldobás előtt ki kell írni a lemezre, mert például módosított lemezlapról vagy cseretárhelyre írandó lapról van szó. Ilyenkor persze fel kell pörgetni a merevlemezt, márpedig éppen ezt szeretnénk elkerülni. ***Andrew Morton*** úgy módosította a memóriakezelőt, hogy az a ***Laptop Mode*** használatakor először kiírást nem kívánó lapokat próbáljon eldobni. Mindezek a módosítások együttesen azt teszik lehetővé, hogy a ***Laptop Mode*** használatakor akár tíz percig is lekapcsolva maradjon a merevlemez. Ha nem túlságosan gyakran módosítunk fájlokat, akkor ennél is hosszabb időszakokat húzhatunk ki a merevlemez használata nélkül. Végül is, ha nincs mit írni, a meghajtót sem szükséges felébreszteni. Sajnos, ha a fájlrendszereket alapértelmezett beállításokkal fűzzük be, akkor a helyzet megváltozik, a fájlrendszer ugyanis rögzíti a hozzáférési időket. A hozzáférési idők akkor is frissítésre kerülnek, ha csak olvassuk a fájlokat, ilyenkor azonban ki kell írni azokat a lemezre. Ezt a kellemetlenséget elkerülendő a ***Laptop Mode*** leválasztja, majd a `mount noatime` kapcsolóját használva fűzi be újra a fájlrendszereket. Ezzel megszűnik a hozzáférési idők rögzítése, és lehetővé válik, hogy hosszabb időtartamot is képesek legyünk eltölteni lemezes be/kivitel nélkül.

Talán már feltűnt, hogy az itt művelt dolgokat – mint a ***/proc*** piszkálása – jellemzően felhasználói térből szokás elvégezni. Nem véletlen, hogy a ***Laptop Mode*** is egy rendszermag összetevőből és egy felhasználói térbeli parancsfájlból áll. A ***Laptop Mode-ot a*** parancsfájl segítségével lehet

© Kiskapu Kft. Minden jog fenntartva

© Kiskapu Kft. Minden jog fenntartva

Tippek és csapdák

MP3-lejátszás

Ha úgy akarunk MP3 fájlokat lejátszani, hogy a merevlemez kikapcsolt állapotban marad, próbáljuk meg növelni a */sbin/laptop_mode READAHEAD* beállításánál megadott értéket. Az is jó megoldás, hogy az összes MP3 fájlt egy kisebb tmpfs RAM-lemezre másoljuk. Ügyeljünk arra, hogy ne legyen túl nagy a RAM-lemez, mert akkor a gép a cseretárhelyre fogja kiírni, és eredeti szándékunkkal pontosan ellentétes eredményt érünk el. Kisméretű RAM-lemezen sok dolgot lehet és érdemes tárolni, ha kikapcsolt állapotban akarjuk tartani a merevlemezt, de természetesen csak akkor, ha az adatok esetleges elvesztése nem okoz gondot.

Egyedi leállási idők

A *Laptop Mode* maximális leállási idejét módosítani is lehet. Ehhez a */sbin/laptop_mode* fájlban szereplő MAX_AGE beállításnak kell azt az értéket adni, ahány másodpercig lemezre íratlanul, a memóriában akarjuk tartani az adatainkat. Az alapérték 600 másodperc, vagyis tíz perc. Amint a *PowerBook-kal* végzett kísérletem eredményéből is látszik, különösebben nem éri meg ennél magasabb értéket megadni. Ha ezt az időt csökkentjük, akkor kevésbé kell aggódnunk adataink elvesztése miatt, de veszítünk nagyjából két perc üzemidőt. Ha valamilyen valóban fontos adatot szeretnénk biztonságban tudni, akkor annak mentése után magunk is elindíthatunk egy szinkronizálást. A *Laptop Mode* figyelembe veszi az ilyen kéréseket, és ilyenkor mindent kiír a lemezre. A szinkronizálás alapállapotba hozza az időzítőket is, vagyis elvégzése után a lemez akár újabb tíz percig is leállítva maradhat.

Leállítás a cpudyn segítségével

Ha cpudynt használunk a processzor órajelének dinamikus szabályozására, talán érdekel bennünket, hogy a program a merevlemez leállítására is képes. Vannak olyan meghajtók, amelyek a tétlenségi időtúllépés beállítását egyszerűen figyelmen kívül hagyják, ha túl alacsonynak találják a kapott értéket. Ráadásul a tétlenségi idő értékét öt másodperces lépésekben lehet beállítani, ezért például nyolc másodpercet nem tudunk megadni. A cpudyn itt jön a képbe, ugyanis a merevlemez közreműködése nélkül figyeli a tétlen időszakokat, és akkor állítja le a merevlemezt, amikor csak akarjuk.

Smart Spindown

A *Laptop Mode* az aktív időszakok végén végez egy szinkronizálást, majd megvárja, hogy a lemez önmagától leálljon. Ugyanakkor jobb, ha a szinkronizálást közvetlenül a meghajtó leállása előtt hajtjuk végre, így ugyanis nagyjából 20 másodperccel frissebb adatokat tudunk kiírni. A *Laptop Mode* erre nem képes, ugyanis nem tud aktív lemezlekapcsolást végezni. *Smart Spindown* névvel írtam egy parancsfájlt, amely képes együttműködni a *Laptop Mode*-dal. Szépnek vagy tökéletesnek véletlenül sem mondanám, de ha minden cseppnyi energiával takarékoskodni akarunk, esetleg az adatok biztonsága szempontjából számít az a húsz másodperc, akkor legalább van mihez nyúlnunk. (Lásd az internetes forrásokat.)

engedélyezni, ez a rendszermag oldali támogatást a */proc/sys/vm/laptop_mode* beállításával kapcsolja be. Ezután leválasztja és újra befűzi a fájlrendszereket, miközben a */proc* bizonyos beállításait is módosítja.

Üzembe helyezés

A ***Laptop Mode*** használatához először is szükségünk van egy a támogatására képes rendszermagra. A ***Laptop Mode a*** 2.6-os ***Linuxokban a*** 2.6.6-os változattól felfelé található meg. A rendszermag forrásfájában a ***Laptop Mode*** leírását a ***Documentation/laptop-mode.txt*** fájlban találjuk. A leírásba beágyazva szerepel a vezérlő parancsfájl, ezt magunknak kell kimásolnunk és elmentenünk `/sbin/laptop_mode` név alatt. Ezután adjunk a parancsfájlra futtatási jogot: `chmod 700 /sbin/laptop_mode`.

A ***Laptop Mode*** engedélyezéséhez rootként a `/sbin/laptop_mode start` parancsot kell kiadni. A parancsfájl minden szükséges műveletet elvégez, kivéve a merevlemez leállítását, amelyhez a meghajtó tétlenségi időtúllépését is meg kell adnunk – erre a `hdparm -S 4 /dev/hda` parancs szolgál. A 4-es érték 20 másodperces tétlenségi időtúllépésre utal. Ha le akarjuk tiltani a ***Laptop Mode-ot***, csak adjuk ki a `/sbin/laptop_mode stop` parancsot.

Célszerű lehet a ***Laptop Mode-ot*** úgy beállítani, hogy akkumulátoros üzemmódra váltáskor azonnal elinduljon. Ha ACPI-támogatással rendelkező gépünk van, akkor tegyük a következőket: másoljuk ki az ***ac_adapter*** és a ***battery.sh*** fájlt a ***Laptop Mode*** leírásából, majd helyezzük őket a megadott helyre. Szerkesszük át a ***battery.sh***-t, adjuk meg benne merevlemezünk eszköznevét és a kívánt tétlenségi időt – és ennyi az egész.

Rejtélyes bekapcsolások

Időnként a merevlemez úgy pörög fel, hogy ennek semmi okát nem látjuk. Ilyenkor meg kell keresni a jelenség hátterét. A ***Laptop Mode*** ismer egy úgynevezett blokk-kiíratómódot is, amelyet a lemezhasználat hibakeresésére lehet alkalmazni. Mielőtt engedélyeznénk, tiltsuk le a ***syslogd***-ben a rendszermag üzeneteinek naplózását, esetleg állítsuk is le teljesen. Ennek módja terjesztésünk felépítésétől függ. Ha nem állítjuk le a ***syslogd***-t, gépünk végtelen ciklusba eshet, ugyanis a hibakeresés kimenetét átveszi a ***syslogd***, majd kiírja lemezre, ám ezzel újabb merevlemezes művelet naplózását váltja ki és így tovább.

A blokk-kiíratómód engedélyezéséhez rootként az `echo 1 > /proc/sys/vm/block_dump` parancsot kell kiadnunk. A rendszermag kimenetében, amelyet most, hogy a ***syslogd*** nem működik, a `dmesg` segítségével olvashatunk el, az alábbiakhoz hasonló üzeneteket fogunk látni:

```
bash(273): READ block 3242 on hda1 (A 3242-es
 ➥blokk olvasása a hda1 eszközről)
bash(273): dirtied inode 10237 (.bash_history) on
 ➥hda1 (A hda1 eszköz 10237-es fájlleírója
 ➥bepiszkolódott)
pdflush(6): WRITE block 3242 on hda1 (írás a hda1
 ➥eszköz 3242-es blokkjába)
```

A kimenet értelmezése a következő. Egy ***bash*** nevű, 273-as azonosítójú folyamat kiolvasta a */dev/hda1* eszköz 3242-es blokkját. Ugyanez a folyamat bepiszkolta a ***.bash_history*** nevű fájlt. A fájl tehát megváltozott, de még nem íródott ki a lemezre. A ***pdflush*** démon ezután írta a 3242-es blokkot, ez alighanem a ***bash*** által korábban módosítottal egyezik meg.

Ha kezünkben a hibakeresési kimenet, megkereshetjük a hiba forrását. Ha *READ* (olvasás) üzenetet látunk, akkor célhoz is értünk. Ki kell találnunk, hogy az adott folyamatnak miért volt szüksége az adatokra, majd eldönthetjük, hogy leállítjuk a folyamatot, vagy megváltoztatjuk az alkalmazás beállításait úgy, hogy többé ne igényelje az adatokat, esetleg olvassa be őket előre, amikor a merevlemez éppen működik. Ha egy fájlt szeretnénk előreolvasni, nem kell mást tennünk, mint hogy kiadjuk a `cat /konyvtar/fajl >/dev/null` parancsot, lehetőleg kétszer is, így a *Linux* csak olvasható alrendszere nem dobja el azonnal a fájlt. Ha csak bepiszkolt fájlra utaló üzenetet látunk, akkor nem kell aggódnunk. Senki és semmi nem ír a lemezre, ezek az üzenetek csak azt jelzik, hogy egy folyamat alkalomadtán kiírást igénylő módosításokat hajt végre. Ilyenkor a merevlemez csak tíz percenként pörög fel, majd rögzíti a változásokat, más nem történik.

Ha tíz percnél gyakrabban látunk *WRITE* üzeneteket, de *READ* üzeneteket nem találunk, vagyis nincs olyan művelet, ami aktiválná a merevlemezt, akkor valószínűleg valamelyik folyamat közvetlenül szinkronizálja az általa kezelt fájlok tartalmát. A *syslogd* például hajlamos ilyesmire. Ha azt látjuk, hogy a *syslogd* nem várt pillanatokban végez írásokat, akkor módosítanunk kell a *syslog.conf* tartalmát. Valószínűleg van benne egy ilyen sor:

```
kern.*      /var/log/kern.log
```

Ez arra utasítja a *syslogd*-t, hogy minden a *kern** maszkkal egyezést mutató naplóüzenet után indítson el egy fsync() hívást. Ha a sort a következőképpen változtatjuk meg:

```
kern.*      -/var/log/kern.log
```

majd újraindítjuk a *syslogd*-t, akkor ezeknél az üzeneteknél többé nem fog `fsync()` hívásokat indítani. Ügyeljünk viszont arra, hogy mely naplófájlokra vonatkozóan végzünk módosításokat. Ha figyelünk a biztonságra, akkor például fontosnak tarthatjuk az *auth.log* szinkronban tartását.

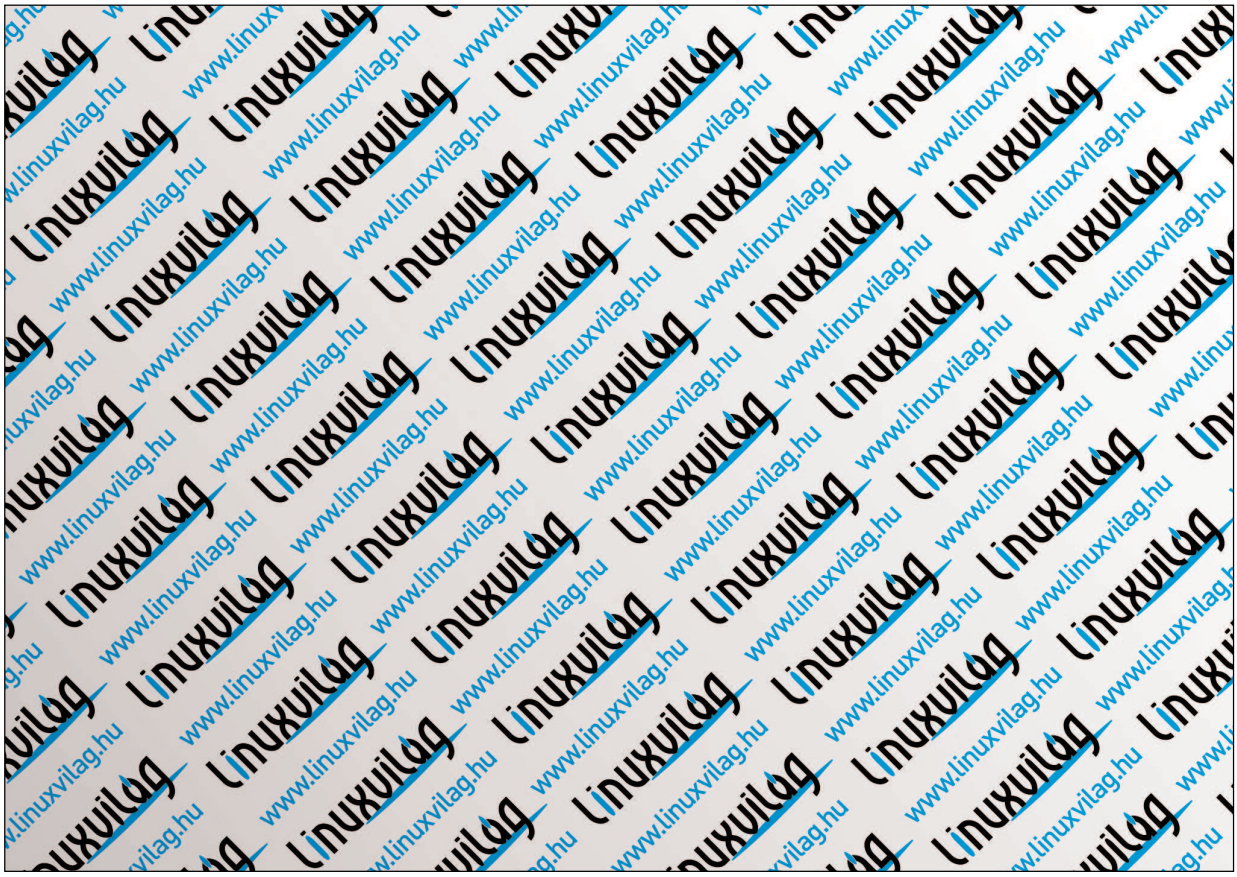
Köszönetnyilvánítás

Ugyan nekem nincs hordozható gépem, mégis rengeteg örömet szerzett nekem a *Laptop Mode-on* végzett munka. Szeretnék köszönetet mondani mindazoknak, akik hozzájárultak a *Laptop Mode* fejlesztéséhez, köztük *Jens Axboenak, Micha Feiginnek, Andrew Mortonnak* és *Kiko Pirisnek*. Ugyancsak hálás vagyok *Jeroen Kruis* segítségéért, aki hagyta, hogy vadonatúj *PowerBook G4*-esét használjam a kísérleteimhez.

Linux Journal 2004. szeptember, 125. szám

Bart Samwel a Laptop Mode az őkezdeményezése nyomán lett a 2.6-os Linux része – noha ő maga nem rendelkezik hordozható géppel. Informatikát tanul a Leideni Egyetemen, Hollandiában, miközben egyedi programok írásából él meg. A ➲ bart@samwel.tk címen érhető el.

© Kiskapu Kft. Minden jog fenntartva



© Kiskapu Kft. Minden jog fenntartva

Linux spektrum
Lycoris te drága...

A Linux és a terjesztések.

Nagyon sok felhasználó számára a Linux azt a terjesztést jelenti, amelyikkel már találkozott. Aztán elkerekedett szemekkel konstatálja a világ sokszínűségét, ha egy másikról is hall. Csak Európában közel 100 különböző összeállítás létezik. Ebből a sokaságból választottam ki egy érdekesnek ígérkező terjesztést. Ezt fogom a cikkben górcső alá venni – és bizony –, kritikával is illetni. Meg fogom vizsgálni alkalmazhatóságát, hibáit, erényeit is.

Lycoris

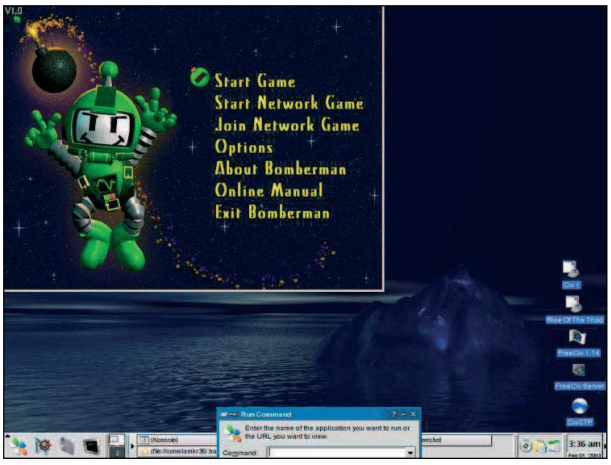
Ez a rendszer egy igazi érdekesség. Azért foglalkozunk vele, mert határozottan meglepő szemlélettel állították össze. Hasonló példa nem sok akad. Szellemiségét tekintve talán a Lindows khm... izé... Linspire áll hozzá a legközelebb. Műszaki szempontból tulajdonképpen nincs benne semmi különös. Ugyanúgy a Linux rendszermagot használja mint a többi terjesztés, és alapértelmezett grafikus ablakkezelője is a megszokott KDE 3.2.3-as. Talán csak a hangulata az, ami teljesen mássá teszi. Nekem a „másik rendszer" jutott róla eszembe. Részben a hibái miatt...

A csomag

A Lycoris a cikk írásának időpontjában még beszerezhető volt, azóta azonban olyan hírek is lábra kaptak, hogy megszűnt. Az én változatomat egy ftp-kiszolgálóról töltöttem le, amely azóta 530-as hibaüzenetet ad, és jelszót kér. Utánanéztem hát a ➲ ***http://www.linuxiso.org*** weboldalon is. Mint kiderült, innen továbbra is működik a letöltés, csak az ftp-re nem enged be.

Az Olvasó most joggal kérdezheti: miért esett a választásom egy olyan rendszerre, aminek a beszerzése ennyire nehézkes. Talán azért, mert valóban figyelemre méltó Linux változatról van szó. Ami viszont teremtőjének üzletpolitikáját illeti, nos az hasonlatossá vált a Linspire-éhez. (Csak halkan jegyzem meg: linuxos körökben az ilyen lépés rendszerint meglehetősen szerencsétlennek számít.)

A Lycoris egyetlen CD-n elfér. Telepítője kifejezetten barátságos és egyszerű, néha talán túlzottan is. Lehet, hogy bennem nem van a hiba, de néha az volt az érzésem, hogy bizonyos dolgokat túlegyszerűsít. Szerettem volna elérni, hogy a telepítési beállításokba egy kicsit belenyúlhassak, de a telepítő ezt nem teszi lehetővé. Kizárólag a saját elképzeléseit követi, és ebbe bele kellett törődnöm, ha működés közben is látni szerettem volna a rendszert. Az is meglepett, hogy ez a terjesztés képes csendben, egyetlen kérdés nélkül legyalulni Master Boot Recordot.



A rendszer a manapság elvárható sebességgel költözött fel a gépre, fél óra múlva már fent is terpeszkedett a merevlemezemen. Az alapvető beállítások megadása után a gép újraindult a telepített rendszerrel. Ekkor következett az első meglepetés. Az új rendszer nem tudott rendesen fölállni, rendszerbetöltés közben több sornyi hibaüzenetet produ-

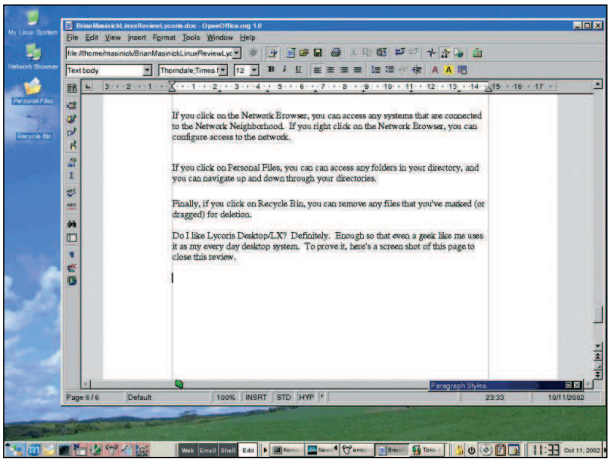
© Kiskapu Kft. Minden jog fenntartva

kált. Azért végül magára talált, de bejelentkezés után hihetetlen lassan működött. Olykor egy percig is eltartott, mire előjött az a menü, amiből újra lehetett indítani a gépet, és a kurzor is akadozva járt. Szerencsére érdemes volt kivárni egy újraindítást, mivel utána már minden tökéletesen működött. A rendszer hibaüzenet nélkül betöltődött és ezúttal úgy tűnt a sebességgel sincs gond.
Kihasználván a lehetőséget természetesen megpróbáltam utánanézni, mi okozhatta a galibát. A naplófájlok körülbelül 30 perces böngészése után (alaposan el vannak dugva) kiderült, hogy az AMD processzorra orrolt meg a Lycoris. Őszintén szólva ezt ma sem értem, hiszen ilyen hibának elvileg nem szabadna felbukkannia. És ha már felbukkant, miért múlt el magától az első újraindítás után? Attól tartok ez már örökre rejtély marad. Szerencsére a macerás kezdés után már minden lehetőségem megvolt, hogy felfedezzem ezt az új világot.

A rendszer

Először természetesen a KDE „Start” menüjét vettem szemügyre. Alaposan végignézve azt kell mondanom, hogy kissé szegényes.
A menüpontok szervezése (és általában a teljes rendszer) a Windowst idézi, sőt, a hasonlóság kifejezetten élethű. Az áttérőknek ez valószínűleg igen hasznos, hiszen az ösztönössé vált mozdulatok itt is működnek. Az persze már más kérdés, hogy ez mennyire hatékony.
A keresés funkció viszonylag jól működik, és teljesen átláthatóak a kategóriák is. Az internet, a multimédia, valamint a hasonló alapszolgáltatások kényelmesen elérhetőek. Jó közelítéssel „fontossági” sorrendben következnek egymás alatt. Az efféle a szaknyelvben szoftver ergonómiának nevezett megfontolások érvényesülését sajnos csak kevés terjesztésnél lehet megfigyelni.
Az egyes kategóriák tartalma ugyanakkor kicsit elkeserített. A Desktop/LX csomag messze nem olyan felszerelt, mint amihez más terjesztéseknél hozzászokhattunk. Amint arra korábban utaltam, a Lycoris sok tekintetben a Windowsra emlékeztet – sajnos az alapszolgáltatások szűkösségében is.
A Windowsra kísértetiesen hasonlító felhasználói felületen könnyen eligazodik az ember. A legfeltűnőbb eltérés talán az, hogy itt a „Sajátgép”-et „My Linux System” -nek hívják. Erre kattintva böngészhetjük a lemezek tartalmát. Kinézetre itt is tökéletes a hasonlóság, bár világosan felismerhető néhány Linuxra jellemző vonás.
Nagyon hasznos a vezérlőpult. Talán mondanom sem kell, hogy ez is tökéletes mása a windowsos beállítóközpontnak, amit ezúttal nyugodtan tekinthetünk pozitívumnak. Viszonylag kevés az olyan terjesztés, amelynek kulturált, kellően részletes, a kezdő felhasználók számára is jól használható kezelőfelülete van. Igaz a Lycoris ezen a téren nem éri el a SuSE `yast2`-jének színvonalát, (ezt sokan a legjobb kezelőfelületnek tartják), de közelít hozzá.
Minden olyan alapbeállítást elvégezhetünk vele, amelyre az internetre való csatlakozáshoz feltétlen szükség van. Ráadásul a rendszer viszonylag gyorsan működik.
Innen érhető el a hálózati frissítés varázsló is. Ha elindítjuk, azonnal kapcsolódik, kigyűjti a frissítéseket, és megkérdezi, folytassa-e ezt a tevékenységet. Ha igennel válaszolunk, a rendszer frissítése már csak a hálózati kapcsolaton múlik. Itt is nehéz eldönteni, hogy a program egyszerűsége bosszantó hiányosság, vagy kifejezett előny. Megvan ugyanaz a betegsége, mint a telepítőnek: túlegyszerűsít, a felhasználónak pedig semmiféle szabadságot nem hagy. Gyakorlatilag egyetlen dolgot lehet benne eldönteni, frissítünk vagy nem. Ebből a szempontból a SuSE frissítőrendszere biztosan veri a Lycorisét.
Az általam tesztelt Personal kiadással egy profi felhasználó sokat nem tud kezdeni. Internetezésre, amolyan vékony kliensnek tökéletes. Talán egy internet kávézóban tudnám leginkább elképzelni, hiszen rendelkezik a Linux előnyeivel (stabil, biztonságos), ugyanakkor szinte a megszólalásig hasonlít a Windowsra. Komolyabb feladatra ugyanakkor alkalmatlan. Otthoni használatra sem tudom nyugodt szívvel ajánlani. A rendszer ugyanis olykor komoly csalódásokat okoz.
Ahhoz lassan kezdek hozzászokni, hogy a fájltársításokkal mindenütt kicsit dolgozni kell, ha az ember tökéletesre vágyik. (A mai -napig nem világos, hogy ha egy terjesztés az `mplayer`-t alapból telepíti, akkor a KDE miért nem ezzel akarja lejátszani a megfelelő tartalmakat.)
A Lycorisnál kicsit súlyosabb a helyzet. Itt időnként nincs is mivel társítani. Az ***mp3***, ***ogg*** és hasonló formátumokhoz ugyan csomagolták az ***xmms***-t, a filmek lejátszása azonban siralmas. Gyakorlatilag csak a KDE lejátszói állnak rendelkezésre, ráadásul a megfelelő kodekek hiányában azok sem ér-

© Kiskapu Kft. Minden jog fenntartva



nek sokat. És ezzel ki is merítettük a Lycoris és a multimédia kapcsolatának tárgyalását. Ez szinte teljesen használhatatlanná teszi a rendszert egy átlagos otthoni felhasználó számára. A bővítések ugyan megrendelhetőek, sőt ezt interneten keresztül is megtehetjük, na de hogy 2004-ben egy Linux terjesztés ne tartalmazza alapból az `mplayer`-t, vagy `xine`-t...

Változatok, bővíthetőség, támogatás

A Lycorisnak két fő változata létezik, a Personal, és a Deluxe kiadás. Mint említettem, én a Personal kiadást teszteltem, amely ingyenesen elérhető. Aki dobozt is szeretne hozzá, annak 40 dollárt kell fizetnie. A Deluxe semmilyen formában nem tölthető le. Kizárólag kereskedelmi úton szerezhető be, viszont ez is csak 50 dollárba kerül. Ez a változat tartalmazza a forrást, és fejlesztői rendszert is.
Minden egyéb csomag külön rendelhető meg, és mindegyikért fizetnünk is kell. A *„Productivity pack”* gyakorlatilag az irodai programcsomagot takarja, és van benne még pár apróság. Ez gyakorlatilag az ***OpenOffice*** irodai programcsomagot jelenti, valamint a PDA szinkronizációhoz használható ***Kpilot*** programot, amelynek egyébként a KDE részének kellene lennie. Ez a csomag további 40 dollárba kerül.
Érdekes még az *„Internet connect”* csomag, amely valószínűleg kiszolgálókat is tartalmazhat, másként nehezen érthető a 229 dolláros ára. Egyedül a *„PowerPack 1.4 perorder”* csomag ára tekinthető ésszerűnek, lévén tartalmazza a ***Crossover Office*** alkalmazást, amellyel ***Microsoft Office*** alkalmazásokat futtathatunk Linuxon.
Kifejezetten figyelemre méltó még a *„Game Pack”* csomag. Habár 35 dollárt kóstál, és csupa egyébként szabadon is használható játékot tartalmaz.
Megtalálható benne a ***WineX*** is, mellyel hirtelen rengeteg játék futtatására lesz képes a Linux. Természetesen minden csomaghoz jár a támogatás, és a kézikönyv, az említett kiegészítők nélkül azonban a Lycoris elég rövid kaland lesz. Annál is inkább, mert szinte semmilyen forrás nem áll rendelkezésünkre. Amit telepítettünk, az úgy is marad.

Hardverigény, futtatás

A kezdeti kompatibilitási gondok leküzdése után, a Lycoris teljesítményben nem volt sokkal rosszabb, mint bármely más terjesztés. A KDE felület eleve sok erőforrást igényel, amit csak tetéz, hogy a Lycoris csapat gondoskodott róla, hogy színes-szagos legyen a felület. Ennek pedig sajnos ára van. A rendszerbetöltés még gyors, a feltuningolt KDE viszont már lomha. Az igazi problémát azonban nem is ez jelenti, hanem az a tény, hogy a Lycorisban a KDE az egyetlen elérhető felület. Ez ismét olyan dolog, amely alaposan korlátozza a felhasználói szabadságot.
A rendszer hardverigénye tehát meglehetősen komoly. 800 Mhz-es Duron processzorral és 512 MB memóriával jól futott ugyan, de nem kényelmesen. Látszott rajta, hogy nem ilyen „kis” gépekre tervezték. Ráadásul a fejlesztők Intel iránti elkötelezettsége is megmutatkozott rögtön az elején.

Összefoglalás

A Lycoris Desktop/LX igen érdekes próbálkozás, és kiváló példa arra, hogy kell a kényelmet a Linux stabilitásával ötvözni. A Windowshoz való hasonlatosságot a készítők sajnos az üzletpolitikára is kiterjesztették, ami szerintem nem szerencsés. A rendszer egy átlagos otthoni felhasználó számára nem túl jó választás, hiszen alapkiépítésben kevés alkalmazást és multimédiás lehetőséget tartalmaz.
Kizárólag olyan helyen tudom elképzelni, ahol a Linux stabilitását és a Windows kényelmét ötvöző„egyengépekre” van szükség. Ilyen lehet egy vékony ügyfél, vagy egy internet kávézó gépparkja.
A cégnek amúgy van egy másik igen érdekes terméke, a Lycoris PDA-ra fejlesztett változata. Ezt sajnos nem tudtam jobban szemügyre venni, mivel kizárólag kereskedelmi úton szerezhető be, ráadásul nekem Palmom van, a rendszer Pocket Pc változata pedig csak a Sharp Zaurus-szal és Compaq Ipaq -al kompatibilis. Ugyanakkor örvendetes, hogy a Lycoris egyáltalán szerepel ezen az egyelőre nem túl széles piacon. Az egyetlen Palm kézigépekre szánt Linux kiadás még gyerekcipőben jár. Eleve csak régi géptípusokat támogat, ami pedig a felszereltségét illeti, programjainak többsége félkész, vagy majdnem teljesen kezdeti stádiumban lévő szoftver.
A Lycoris tehát megmarad érdekességnek. Igazán reális, és figyelemre méltó fejlesztésnek egyedül a PDA-s változat ígérkezik. Ezzel valóban öregbítheti ez a cég a Linux nevet.

Dancsok „strogg” Zoltán

© Kiskapu Kft. Minden jog fenntartva

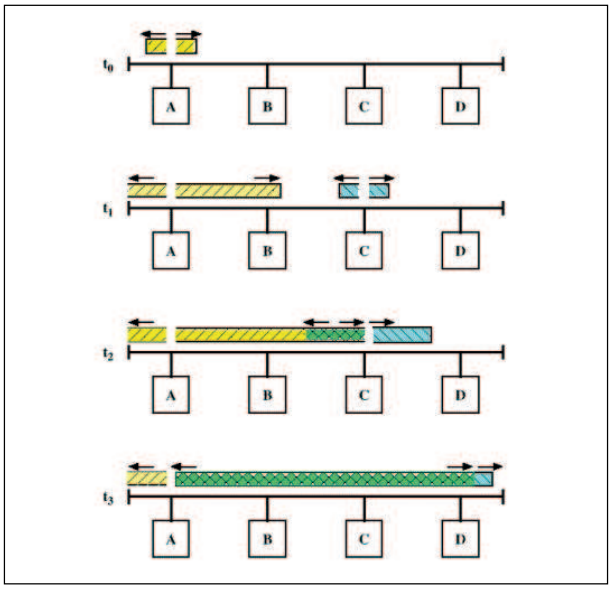
Hálózatok (10. rész)
A közegelérési alréteg, az ALOHA, a CSMA, és az Ethernet

Ebben a részben még mindig az adatkapcsolati réteget taglaljuk, és ezen belül ennek egy alrétegét a közegelérési alréteget vesszük szemügyre. Ez a LAN-ok és a műholdas hálózatok esetében játszik kulcsszerepet.

Sokféle hálózatot hord hátán a Föld, ám ezek mindegyike két nagy csoportra osztható. Az előző részben e két nagy hálózati-osztály egyikével foglalkoztunk, mégpedig azokkal, amelyek úgynevezett kétpontos összeköttetést használnak. Ebben az esetben a kommunikációban résztvevő két fél egymással egy zárt csatorna segítségével van összekötve.
A hálózatok másik típusát az adatszóró hálózatok jelentik, amely esetén egy csatornán egyszerre több állomás is osztozik. Ez nagymértékben hasonlít egy telefonos konferenciabeszélgetéshez, ahol minden résztvevő saját telefonkészülékkel rendelkezik, és a résztvevők mindnyájan hallják egymást. Az adatszóró hálózatok megvalósításakor a legfontosabb kérdés az, hogy versenyhelyzet kialakulása esetén mi legyen a teendő. Versenyhelyzetről egyébként akkor beszélünk, amikor egy erőforráshoz egyszerre többen is hozzá szeretnének férni. Ha nincs korlátozás, és mindenki akkor veszi igénybe az erőforrást amikor csak akarja, akkor borzalmas dolgok történhetnek. Térjünk vissza a példánkhoz: ha az említett konferencián egyszerre többen kezdenek el beszélni, akkor a többiek csak érthetetlen hangzavart hallanak. Kézenfekvő tehát, hogy szükség van egy olyan protokollra, amely versenyhelyzet esetén eldönti, hogy a csatornára igényt tartók közül ki nyerje el a jogot az adásra.
Ezzel a problémával az adatkapcsolati réteg egy alrétege, a ***közegelérési alréteg (MAC, Medium Access Control)*** foglalkozik, a következőekben ezzel fogunk foglalkozni. Jelen cikkben az állomások közötti csatorna-kiosztás módszereivel fogunk megismerkedni.

Csatornakiosztásos módszerek

A versenyhelyzeteket a legegyszerűbben úgy kerülhetjük el, ha előre meghatározott időközönként a csatornát másmás állomás használja. Ezt hívjuk ***TDM-nek (Time Division Multiplexing, időosztásos nyalábolás)***. Azt a rövid időintervallumot, amely alatt egy állomás jogosult az adásra, az adott állomás időrésének nevezzük. Amikor egy olyan állomásra kerül a sor, amelynek éppen nincs mondandója, akkor az ő időrése kihasználatlan marad. Ehhez hasonló megközelítés az ***FDM (Frequency Division Multiplexing, frekvenciaosztásos nyalábolás)***, azzal az eltéréssel, hogy itt az idő helyet a csatorna sávszélességét osztják fel annyi részre, amennyi állomás kapcsolódik a csatornára.
Sem az ***FDM***, sem a ***TDM*** nem jó megoldás abban az esetben, amikor a csatornára kapcsolt állomások száma nem állandó, hanem folyamatosan változik. Ha példának okért a sávszélességet N egyenlő részre osztjuk, és egy adott pillanatban N-nél kevesebb számú állomás szeretne kommunikálni, akkor a sávszélesség egy része kihasználatlan marad. Súlyosabb a probléma akkor, ha N-nél több felhasználó szeretne dolgozni a hálózaton. Ilyenkor elő fog fordul, hogy valaki számára nem marad szabad sáv, és ebben az esetben akkor sem lesz képes forgalmazni, ha esetleg egyes állomá-



1. ábra

© Kiskapu Kft. Minden jog fenntartva

sok éppen nem használják a csatornát. Az ***FDM*** (és ugyanez igaz a ***TDM***-re is) akkor sem lehet hatékony megoldás, ha minden időben N darab állomás van a csatornára kötve. Ugyanis ha egy állomás nem használja a csatornát, akkor az ő sávja kihasználatlanul marad, mivel egyik állomás sem veheti igénybe a másikhoz tartozó frekvenciatartományt. A legnagyobb visszaesés a hatékonyság terén akkor tapasztalható, amikor az információ áramlása löketes jellegű (a maximálisan forgalmazható adat mennyisége jóval nagyobb az átlagosnál), mivel az egyes frekvenciasávok legtöbbször kihasználatlanul maradnak.

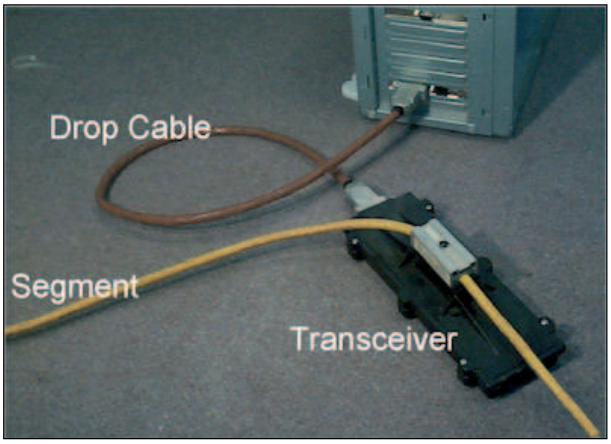
A 70-es években kidolgoztak egy akkor új megoldást a csatorna kiosztásának problémájára. Ez a fejlesztés az ***ALOHA*** nevet kapta, amelyet eredetileg rádiós üzenetszórásos rendszerekhez alkottak meg, de remekül működik minden megosztott csatornán.

Az ***ALOHA*** alapötlete az, hogy minden állomás bármikor, bármennyit adhat. Ha két állomás egyszerre kezdi ezt el, akkor úgynevezett ***ütközés*** lép fel, és az állomások hibás kereteket fognak kapni. Minden állomás képes észlelni az ütközést, mivel a keretek ellenőrzőszáma helytelen lesz. Ha a küldő állomás azt tapasztalja, hogy a keret elküldésével egy időben más is adott, akkor véletlenszerű ideig várakozik, majd megkísérli ismét forgalmazni a szóban forgó keretet. Jogosan merül fel a kérdés, hogy mennyire lehet hatékony ez a módszer. Tegyük fel, hogy minden keret ugyanolyan méretű. Egy keret akkor nem fog elszenvedni ütközést, ha más állomás nem próbálkozik vele egyszerre egy másik keret elküldésével. Ha egy keret elküldése `t` időbe telik, és a keretet a `t0` időpontban bocsátottuk a csatornára, akkor egyik állomásnak sem szabad a `t0 + t` időpontig adni. Egy keret sértetlenségének másik fontos előfeltétele az, hogy a küldés pillanatában ne legyen más keret a csatornán. Mivel minden állomás bármikor adhat (anélkül, hogy megnézné, van-e forgalom a csatornán), ezért a keret akkor is megsérül, ha bármely más állomás adott a `t0 - t` és `t0 + t` időintervallumban. Különböző valószínűségszámítási módszerekkel kiszámolható, hogy az ***ALOHA*** csatorna-kihasználtsága 18%. Ez valóban elég soványnak tűnik, de ahhoz képest nem olyan rossz az eredmény, hogy az állomásoknak egyébiránt nem kell egymáshoz alkalmazkodniuk, mindenki akkor ad, amikor csak szeretne.

Ez a csatorna-kihasználtsági arány megduplázható úgy, hogy az időt felosztjuk egyenlő szakaszokra, amelyek mérete megegyezik egy keret elküldési idejével. Ezután az állomások kizárólag egy ilyen időszakasz kezdetén küldhetnek kereteket a csatornára. Ehhez persze az kell, hogy minden állomás szinkronban legyen a többivel, azaz előre megállapodjanak a szakaszhatárok helyében. Erre az egyik lehetséges módszer az, hogy egy előre kijelölt állomás az egyik időszakasz kezdetén speciális jelet küld a csatornára. Ezt a többiek érzékelik és ehhez hangolják saját órájukat. Ez az úgynevezett ***réselt ALOHA***.

Csatornafigyelő protokollok

Nehéz az ***ALOHA-nál*** jobb megoldást találni abban az esetben, amikor az állomások nem tudják ellenőrizni, hogy éppen forgalmaz-e valaki más is a csatornán. A ***LAN-ok*** esetében az egyes állomások azonban érzékelik a forgalmat, ezért lehetőség kínálkozik olyan protokollok használatára



2. ábra

is, amelyek tekintettel vannak a többi állomás működésére is. Ezeket nevezzük ***csatornafigyelő protokolloknak***, amelyek – nem meglepő módon – jóval hatékonyabbak az ***ALOHA-nál***. Az állomások azon képességét, amely segítségével érzékelni tudják, hogy éppen foglalt-e a csatorna, ***vivőjel-érzékelésnek*** nevezzük.

A legegyszerűbb csatornafigyelő protokoll az 1-perzisztens CSMA ***(Carrier Sense Multiple Access, vivőjel-érzékelés többszörös hozzáféréssel)***. Működése egyszerű: ha egy állomás küldeni akar, de a csatorna foglalt, akkor addig vár, amíg az fel nem szabadul. Amint ez bekövetkezik, az állomás útnak indítja a keretet. Ha ütközés lép fel, akkor véletlen ideig várakoznia kell, majd ismét meg kell próbálnia elküldeni a keretet. Az „1-perzisztens" kifejezés azt jelenti, hogy amint felszabadul a csatorna, az adásra kész állomás pontosan 1 valószínűséggel (azaz biztosan) fog keretet küldeni. Láthatjuk, hogy ez egy meglehetősen türelmetlen protokoll. Eme tulajdonsága némiképp csökkenti a csatorna hatékony kihasználását. Ha ugyanis két állomás is adásra kész állapotban van, akkor amint felszabadul a csatorna, mindketten egyszerre kezdenek el adni. Ez értelemszerűen keretütközéshez vezet.

Ennek kiküszöbölésére megoldás lehet a ***nemperzisztens CSMA*** használata. Ez már egy kicsit türelmesebb protokoll, azaz ha a csatorna foglalt, nem figyeli folyamatosan, hogy az mikor szabadul már fel, hanem véletlen ideig vár. Ha ekkor már szabad a csatorna, akkor adni kezd, ha nem, akkor valamennyit ismét várakozik. Ezzel a módszerrel valóban nő a csatorna kihasználtsága, viszont ennek ára van: az állomás nagyobb késleltetésekkel küldi a kereteket.

Jobb megoldást kínál a ***p-perzisztens CSMA*** protokoll. Hasonlóan a réselt ***ALOHA-hoz***, itt is csak meghatározott időközönként szabad kereteket küldeni. Ha valaki adni szeretne, akkor először bele kell hallgatnia a csatornába, hogy van-e rajta forgalom. Ha nincs, akkor `p` valószínűséggel adni fog, vagy `q = p - 1` valószínűséggel várakozik a következő időrés kezdetéig.

Fokozhatjuk a hatékonyságot azzal, hogy az adó állomásokat képessé tesszük a keretek ütközésének érzékelésére. Erre a legjobb módszer a csatornán észlelhető jelek impulzushosszának és feszültségszintjének figyelése, ezeket utána össze kell hasonlítani a kibocsátott jellel. Ha nem egyeznek,

3. ábra

akkor minden bizonnyal keretek ütközéséről van szó. Ilyenkor fel kell függeszteni a keret küldését, majd az adás véletlen idejű várakozás után kísérelhető meg újból. Ezzel a módszerrel működik a ***CSMA/CD (CSMA with Collision Detection, CSMA ütközésérzékeléssel)*** protokoll, amely időt takarít meg azzal, hogy a biztosan sérült keretek küldésének idejét megspórolja.

Vessünk egy pillantást az *1. ábrára*, amely a ***CSMA/CD*** működését mutatja be akkor, amikor két állomás időben egyszerre ad. A `t0` időpillanatban még csak az A állomás kezdi el a keret küldését. A `t1` időpillanat annyira közel van a `t0`-hoz, hogy a kicsit távolabb lévő C állomás még nem érzékelhette, hogy az A adni kezdett, így ő a csatornát még szabadnak érzékeli. A `t2` időpontban a C állomáshoz megérkeznek az A által küldött keret első bitjei. Mivel ezek „összekeveredtek" a C által küldött bitekkel, ezért ütközés áll fenn, így a C azonnal befejezi a keret küldését. Az A ekkor még tovább küld, mivel ő még nem érzékelte az ütközést. Kicsit később az A is észleli, hogy nem az van a csatornán, amit ő küldött, így azonnal be is fejezi az adást.

Az IEEE 802.3 szabvány

Ez egy 1-perzisztens ***CSMA/CD*** protokollt definiáló szabvány, amely a legelterjedtebb a ***LAN-ok*** világában. Az első ilyen hálózatot a ***Xerox*** építette, amely egy 1 km-es kábel segítségével több mint száz állomást kapcsolt össze, és mintegy 2,49 Mb/s-os sebességre volt képes. Ezt a rendszert ***Ethernet-nek*** nevezték el. (Ez az megnevezés a luminiferous éter kifejezésből ered, amely a XIX. századi fizikusok vélekedése szerint kitölti az egész teret, és lehetővé teszi az elektromágneses hullámok terjedését).

Az ***Ethernet*** hamar sikeres lett, és a ***Xerox*** sok más céggel együttműködve (például az ***Intel-el*** és a ***DEC-el***) létrehozta a 10 Mb/s-os ***Ethernet*** szabványt. Ez 1983-ban történt, azonban azóta sokat fejlődött a technika. 1995-ben egy új ***Ethernet*** szabványt vezettek be, a ***Fast Ethernet-nek*** nevezettet, amely már 100 Mb/s-os átviteli sebességre is képes volt. A ***Fast Ethernet-re a*** későbbiekben még részletesen kitérünk.

Ethernet ábelezés

Maga az „ether", vagy magyarul éter szó a kábelezésre utal, ezért hamarjában nézzük is meg, milyen módon köthetjük számítógépeinket egy ***Ethernet*** hálózatba. Az első kábeltípus a ***10Base5***, avagy polgári nevén ***vastag Ethernet***, amely valóban egy igen vastag koaxiális kábel. Előnye, hogy egy kábelre akár száz gépet is felfűzhetünk, és meglehetősen hosszú, akár 500 méteres szegmenseket is létrehozhatunk. Ebből kifolyólag ez a típusú megoldás leginkább gerinchálózatnak alkalmas. Hátránya azonban, hogy drága, és nehézkes telepíteni. A ***2. ábrán*** láthatjuk, hogy miként kapcsolhatjuk az állomásokat egy ilyen kábelre. Ehhez egy adó-vevőre ***(transceiver)*** van szükségünk, amely a kábelhez csatlakozik, mégpedig úgy, hogy egy tüskét mélyeszt a kábel fém magjába (szokás ezért vámpír-csatlakozónak is hívni). Az adó-vevő feladata a vivőjel érzékelése, illetve a keret-ütközések felfedezése.

Ezt az eszközt az állomással egy másik kábel köti össze, amely a számítógépbe szerelt vezérlő-kártyához csatlakozik. Ennek az áramkör arra való, hogy az adatkapcsolati réteg számos feladatát ellássa: kereteket kell küldenie és fogadnia, ellenőrző összegeket kell kiszámítania, stb.

A következő kábeltípus a ***vékony Ethernet***, vagy hivatalos nevén a ***10Base2***. Ez az előbbinél egy jóval vékonyabb koaxiális kábel, ezért egyrészt olcsó, másrészt könnyebben hajlítható. Ebből adódóan könnyebben el lehet vezetni például a fal mellett. Hátránya viszont a ***10Base5***-höz képest az, hogy legfeljebb csak 200 méter hosszú szegmensek kialakítására alkalmas, és egy ilyen szegmens nem több, mint 30 állomást tartalmazhat.

Az állomások bekötése viszont egyszerűbbé vált. A kábelt egy nem túlzottan bonyolult, ***T alakú BNC-nek*** nevezett csatlakozóval kapcsolhatjuk a géphez. A vivőjelet és az ütközést érzékelő elektronika így az állomás ***(hálózati) illesztő***-kártyájára került. Ennek köszönhetően az ilyen kábelezésű hálózatok telepítése, illetve újrakonfigurálása sokkalta egyszerűbb feladat, mint az a ***10Base5*** esetében.

Mindkét kábelezésnek van azonban egy súlyos hiányossága: nehéz a hibakeresés. Ha a kábel valahol megtörik, vagy egy csatlakozó meglazul, akkor egyrészt sorban ellenőrizni kell az összes állomást, másrészt a hiba az összes állomásra kihat. Ez azt jelenti, amíg meg nem találjuk a probléma forrását, addig az egész hálózatunk használhatatlan állapotban van. Ennek kiküszöbölésére született meg a ***10Base-T*** elnevezésű kábelezés, ahol minden géptől egy külön csavart érpárú kábel vezet az úgynevezett ***elosztóhoz (hubhoz).*** Talán

© Kiskapu Kft. Minden jog fenntartva

Szavazz a CD-mellékletről!

Tavasszal „Szerkeszd te is a Linuxvilágot!" felhívással egy on-line kérdőív kitöltésére kértük olvasóinkat honlapunkon, melynek értékelése a júliusi számban jelent meg (a bővebb változat honlapunkon is elérhető http://www.linuxvilag.hu/hir/1022/711.html). Az eredmény alapján készítettünk egy tervezetet a CD-melléketre vonatkozó változtatásokra. Ennek megvalósításáról a Ti szavazataitok fognak dönteni, ezért kérünk mindenkit, hogy válaszoljon **3 kérdésre** ezen az oldalon:

http://www.linuxvilag.hu/kerdoiv_cd

© Kiskapu Kft. Minden jog fenntartva

nem kell ecsetelnünk, hogy ez a megoldás mennyire megkönnyíti új állomások telepítését, illetve a hálózatból régiek eltávolítását. Hátránya az, hogy a ***hub*** és az állomások között a maximális távolság csak 100, esetleg 150 méter lehet. Ezenkívül az elosztók ára sem mindig a legbarátságosabb. Ennek ellenére ma már ez a kábelezési forma a legnépszerűbb.
A 100 Mb/s-os sebességű ***Fast Ethernet*** hálózatok is ezt használják, igaz, ennek egy gyorsabb változatát, a ***100Base-T-t.***
A negyedik és egyben utolsóként említett kábelezési lehetőség a ***10Base-F***, amely üvegszálakat használ az állomások összekapcsolásához. Ez rendkívül drága megoldás, ugyanakkor a nagy zajtűrés miatt hihetetlenül jó minőséget biztosít. Abban az esetben érdemes ezt használni, amikor viszonylag nagy távolságot szeretnénk áthidalni. Példának vehetjük azt az esetet, amikor az összekötni kívánt gépek két külön épületben vannak.
Akármelyik kábelezési eljárást is alkalmazzuk, a távolságok behatároltak lesznek. Ha valami oknál fogva arra volna szükség, hogy a megengedettnél nagyobb kiterjedésű LAN-t építsünk, akkor úgynevezett ***ismétlőkre (repeaters)*** lesz szükségünk, amelyek felerősítik a rajtuk átmenő jeleket. Fontos lehet tudni, hogy ezek az eszközök a fizikai réteg szintjén dolgoznak, azaz nem tesznek mást, mint minden irányban megismétlik az általuk kapott a jeleket. Vannak olyan eszközök, amelyek képesek a „magasabb" szinten lévő struktúrákkal is foglalkozni, például a keretekkel vagy a csomagokkal. Ilyen például a ***híd*** vagy az ***útválasztó (router)***, amely eszközökkel a későbbiekben részletesen is foglalkozunk.

Az Ethernet MAC protokollja

Nézzük meg felszínében, a gyakorlat során miként is működik az ***Ethernet***. Kezdjük a keretek felépítésének bemutatásával! A ***3. ábrán*** láthatunk erre egy példát. Szürkével jelöltük az úgynevezett előtagot, amelynek értéke mindig 10101010. (Nem véletlen, hogy az előtagban mindegyik bit különbözik a szomszédaitól. A ***802.3*** szabványok úgynevezett ***Manchester*** kódolást alkalmaznak, amely segítségével az állomások meg tudják különböztetni a logikai 0-t a csatorna forgalom-mentes állapotától. Az ilyen minta ***Manchester*** kódolása lehetővé teszi, hogy a vevő az óráját az adóéhoz szinkronizálja). Ezután a keret kezdetét kijelölő bájt következik.
A következő két mező a cél, illetve a forrás cím, amelyek 48 bitből állnak. Lehetőség van arra, hogy egy keretet ne csak egy gépnek címezzünk. Erre szolgál a cél címének legfelső helyiértékű bitje, amelynek ha 1 az értéke, akkor a célcím csoportcímként értelmezhető. Ilyenkor a keret minden olyan állomáshoz eljut, amely tagja a kérdéses csoportnak. Ez az úgynevezett ***többes küldés (multicasting)***. Lehetőség kínálkozik arra is, hogy a hálózatba tartozó összes géphez elküldjük a keretet. Ilyenkor beszélünk ***adatszórásról (broadcasting)***, és ebben az esetben a célcím minden bitjét 1-re kell állítanunk.
A címbitek közül mindenképp ki kell még emelnünk a 46. bitet, amely az előbb tárgyalt legmagasabb helyiértékű bit szomszédja. Ez mondja meg ugyanis, hogy a kérdéses cím helyi vagy globális. A helyi címek értelemszerűen csak az adott hálózaton belül érvényesek, és ezeket a hálózatot üzemeltető személyek jelölhetik ki. Más a helyzet a globális címek esetén, mivel ezeket egy központi szervezet ***(az IEEE)*** határozza meg, méghozzá úgy, hogy a világ összes hálózatikártyája egyedi ***MAC*** címmel rendelkezzen.
Ezután következik az adatmező hosszát tartalmazó mező. Egy ***Ethernet*** keret legfeljebb 1500 bájtnyi adatot tartalmazhat. Az adatmező méretének elméletileg nincs alsó korlátja, de a 0 hosszúságú adatmezők a gyakorlatban két ok miatt mégsem megengedettek.
Az első ok az, hogy amikor az állomás adó-vevője keret-ütközést érzékel, nem küldi el a keret hátralévő részeit. Így a kábelen sok félig elküldött keret lesz jelen. Mivel ezeket a csonka kereteket valahogy meg kell különböztetni a sértetlenektől, ezért a szabvány azt írja elő, hogy minden keretnek legalább 64 bájt hosszúnak kell lennie. Ha ebből levonjuk a keret többi részének méretét, akkor azt kapjuk, hogy legalább 46 bájtnyi adatot kell minden keretnek tartalmaznia. Ha mégis kevesebb volna az elküldendő adat, akkor egy úgynevezett töltelék mezőt kell bevezetni az adatmező után (az ábrán ezt nem tüntettem fel).
A túlságosan rövid keretek tiltásának másik oka ennél sokkalta nyomósabb. Ha a ***LAN-unk*** elég nagy, akkor előfordulhat, hogy egy 64 bájtnál rövidebb keret küldése előbb befejeződik, mint ahogy annak eleje a hálózat legtávolabbi részéhez elérne. Ha a hálózat „határvidékén" egy állomás szintén egy keret küldésével próbálkozik (mit sem sejtve arról, hogy már úton van felé egy másik), keret-ütközés fog fellépni. Ezt érzékelni is fogja, és ennek következtében meg is szakítja saját keretének küldését. A probléma azonban az, hogy a többi állomás is érzékelni fogja a keret-ütközést, így egyik keret sem fog célba érni. A rövid keretet küldő állomás azonban az ütközésről csak a saját keretének elküldése után értesül, azaz azt fogja feltételezni, hogy az sértetlenül megérkezett.
Mivel az ilyen helyzetek téves működést eredményezhetnek, ezért kiszámolták, hogy a szabványban meghatározott maximális méretű hálózaton (2500 méter, 4 ismétlővel), mekkorának kell lennie a legkisebb keret méretének, hogy ne jelentkezhessen a fentebb említett probléma. Arra az eredményre jutottak, hogy egy keret elküldésének legalább 51,2 μs-ig kell tartania. Ez pontosan 64 bájt útrabocsátásához elegendő idő.
Fontos megjegyeznünk, hogy ez az érték csak a 10 Mb/s-os ***Ethernet*** hálózatok esetében ennyi. Ha a hálózat sebessége nő, akkor nagyobb minimális keretméretet kell megnövelni, vagy a maximális kábelhosszúságot szükséges csökkenteni. Összehasonlításként megemlíthetjük, hogy egy 2500 méteres 1 Gb/s-os sebességű hálózat esetén a legkisebb megengedett keret méretét 6400 bájtra kell növelni.
Az ***Ethernet*** keretek legutolsó mezője az ellenőrzőösszeget tartalmazza, amely a keret sértetlenségéről tanúskodik, vagy éppenséggel jelzi a zavart. Ennek segítségével tudjuk kiszűrni a vezetéken keletkező zajok okozta átviteli hibákat. Az ellenőrzőösszeg kiszámításához a ***CRC*** algoritmust használják.
A következő részben tovább taglaljuk az ***Ethernet*** hálózatok felépítését. Szó esik még a ***hidakról*** és a nagy sebességű hálózatokról is. Ezután megismerkedünk az egyetlen olyan WAN hálózattal, amelyik szintén adatszóró csatornát használ: a műholdas hálózattal.

Garzó András

© Kiskapu Kft. Minden jog fenntartva

A drótnélküli konyha

Avagy hogyan kezeljük rádiós hálózati beállításainkat mozgás közben, és miként találjuk meg a legjobb minőségű kapcsolatot.

Nem *Francois*, köszönöm, most nem akarok leülni. Igen, és is látom, hogy az asztalok üresek. Ez azonban csak azért van, mert a vendégeink még nem érkeztek meg. Hogyan? Á értem már! Azon csodálkozol, hogy körbe-körbe járok itt kezemben ezzel a laptoppal és közben egy pillanatra se ülök le sehova. Nos, kedvesem, az indok igen egyszerű. Rádiós hozzáférési pontokat telepítettem a borospincénk minden zugába, amelyeket természetesen vendégeink is használhatnak. Most azért járok itt föl-alá a laptoppal, hogy ellenőrizzem a jel erősségét. Meg akarok győződni róla, hogy vendégeinknek mindenütt lesz kapcsolata a külvilággal, akármelyik asztalhoz is ülnek le.
Isten hozta önöket kedves vendégeim *Marcelnél*, a legfinomabb linuxos konyhában, és a legjobb borok hazájában, ahol immár vezeték nélküli internet kapcsolatot is kiépítettünk. Kérem foglaljanak, helyet és helyezzék magukat kényelembe. *Fancois*, irány a pince, és hozd fel nekünk az 1998-as évjáratú Édenvölgyi Ausztrál Rizlinget!
Mielőtt önök megérkeztek, kedves vendégeim, hosszasan vizsgáltam, vajon az étterem minden pontjáról azonos minőségben érhető-e el a vezeték nélküli kapcsolat. Amíg hagyományos hálózati csatolókártyákat használtunk, egy kis lámpa világított, ha volt élő kapcsolatunk. A rádiós kapcsolatot biztosító kártyáknál a helyzet már korántsem ilyen egyértelmű. A szolgáltatás minőségét is a hozzáférési ponttól való távolság, a jelerősség, a jelfogó elhelyezése és egyéb ehhez hasonló tényezők határozzák meg. Mármost nincs is ezzel semmi baj, ha az ezzel kapcsolatos információk a rendelkezésünkre állnak.
A napjainkban rendelkezésre álló segédeszközök egész sor lehetőséget biztosítanak a jelerősség méréséhez, vagy az elérhető hozzáférési pontok megtalálásához. A legtöbb ezek közül erősen támaszkodik *Jean Tourrilhes* munkájára, ki a rendszermagba segített beépíteni a vezeték nélküli kapcsolatok támogatását. Én is megállok tehát most, hogy munkája előtt tisztelegjek. Látom *Fancois* már felszolgálta a borokat, tehát itt az ideje, hogy belekóstoljunk a mai menü első fogásába.
Amint azt már többször említettem, kifejezetten odavagyok a *WindowMaker* alatt futó dokkoló alkalmazásokért, amelyek olyan egyszerűek és olyan keveset foglalnak el a felhasználói felületből. Most is három ilyennel fogom kezdeni a bemutatót. Az első *Jess Mahan WmWiFi* nevű alkalmazása (lásd az *1. ábrát*). Ezen a kis *WindowMaker* alkalmazáson azt

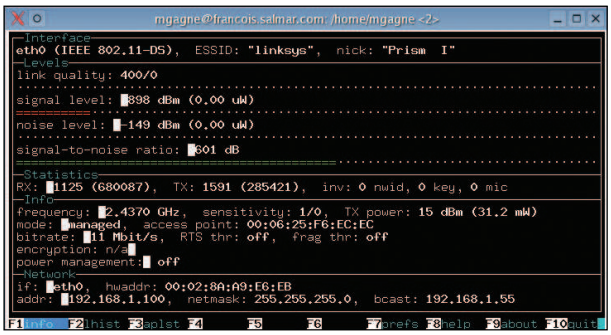
1. ábra A WmWiFi, a wminfo és a wmWave felülete

hiszem azért akadt meg a szemem, mert erősen hasonlít az első mobiltelefonom térerőt jelző alkalmatosságára. Letölthetjük Debian csomag formájában, de természetesen rendelkezésre áll a forrás kód is. Utóbbi lefordítása a már jól ismert öt lépésből áll:

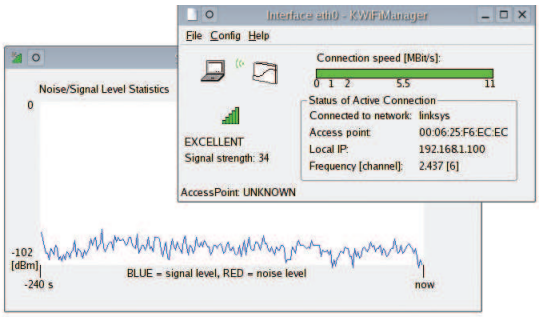
```
tar -xzvf wmwifi-0.4.tar.gz
cd wmwifi-0.4
./configure
make
su -c "make install"
```

Futtatásához a parancssorban adjuk ki a `wmwifi` parancsot. Ha nem *WindowMaker* ablakkezelőt használunk, a kép esetleg nehezen jelenik meg. Ilyenkor lehet hasznos a `wmwifi -bw` formában való indítás, amelynél a megadott kapcsoló „nem megfelelő ablakkezelő" (broken window manager) rövidítése. (Nem állíthatom, hogy a szerzőnek nincs humora...) A *WmWiFi* alapértelmezett megjelenése a szürke háttér, de ha a dokkolt alkalmazásra kattintunk, bekapcsolódik a háttérvilágítás, amitől az LCD színe átvált szép zöldre.
Egészen hasonló a *Carsten Schürmann* által készített *wmWAVE*, amely a térerő mérése mellet néhány egyéb szolgáltatást is kínál. (A programot immár *Jens Schürmann* tartja karban.) Ez az alkalmazás kijelzi a kapcsolat minőségét, illetve a zajszintet is. A forráskód lefordításához egyszerűen csomagoljuk ki fájlokat, majd adjuk ki a `make` és `make install` parancsokat. Futtatásához a parancssorban a `wmwave` parancsot kell kiadni.
A harmadik *WindowsMaker* alkalmazás az *Ico Doornekamp* által készített *wmifinfo*. szemben az előzőekkel ez nem egy kifejezetten vezeték nélküli alkalmazás. Elindításakor körbenéz, detektálja az össze hálózati kapcsolatot, és valamennyiről információt szolgáltat. Ha rádiós kapcsolatot ta-

© Kiskapu Kft. Minden jog fenntartva



2. ábra A wavemon sokmindent megmutat



3. ábra KWiFiManager

lál, akkor kijelzi a mért térerőt. Forráskódjának elérhetőségét a cikkhez tartozó források felsorolásában találjuk. Lefordításához megint nem kell mást tennünk, mint kicsomagolni, majd kiadni a `make` és a `make install` parancsokat. Futtatásához a `wmifinfo` parancsot kell használnunk. Aki ragaszkodik a szürke LCD panelhez, az használja a `-l` kapcsolót is. Ha a gépben több hálózati interfész is van, azok között a jobb egérgombbal válthatunk. Végül a `-i` kapcsoló segítségével megadhatjuk, hogy a program elinduláskor melyik csatoló adatait jelezze ki (például `wmifinfo -i eth0`).

A parancssor szerelmesei, akárcsak jómagam, szeretik az olyan alkalmazásokat, amelyek grafikus felület nélkül, de mégis stílusosan látnak el egy feladatot. ***Jan Morgenstern wavemon*** nevű programja ebbe a csoportba tartozik. Ez az alkalmazás egyetlen ***ncurses*** ablakban jeleníti meg a hálózati kapcsolatok minőségét jellemző összes paramétert. A minőséget dinamikus oszlopgrafikon jelzi. A ***wavemon***-nak ezen kívül van hisztogramot megjelenítő nézete, illetve lehetővé teszi a beállításoknak a képernyőn történő megadását is (lásd a *2. ábrát*).

A ***wavemon*** fordítása ismét csak a szokásos öt lépésből álló folyamat, futtatásához pedig a `wavemon` parancsot kell használnunk. Az induló képernyőn összefoglaló adatokat láthatunk, amely magában foglalja a jelszinteket, a TCP statisztikát, a működési mód leírását, a titkosítást, a bitsebességet, az aktuális hozzáférési pont leírását, a helyi hálózati kártya adatait és sok egyebet. Valamennyi műveletet a funkcióbillentyűk segítségével indíthatjuk. A hisztogram nézet bekapcsolásához nyomjuk meg az F2-t, az áttekintő nézethez pedig az F1 segítségével térhetünk vissza. A beállító képernyőhöz az F7-et megnyomva férhetünk hozzá. Ha netán elfelejtenénk, hogy melyik gomb mire való, csak vessünk egy pillantást a képernyő aljára, ahol mindig látható az összefoglaló.

A ***KDE*** legújabb, 3.2-es verziószámot viselő változatában már megtalálhatunk egy ***KWiFiManager*** nevű, igen tetszetős alkalmazást is. Ez a program, amit ***Stefan Winter*** írt, az előzőekhez hasonlóan képes kijelezni a rádiós kapcsolatok minőségét, de ezen túl számos egyéb szolgáltatása is van. Lehetővé teszi például, hogy négy különböző konfigurációt állítsunk be. Azok az „országúti harcosok”, akik életüket irodáról irodára vándorolva töltik, igen hasznosnak találják majd ezt a lehetőséget.

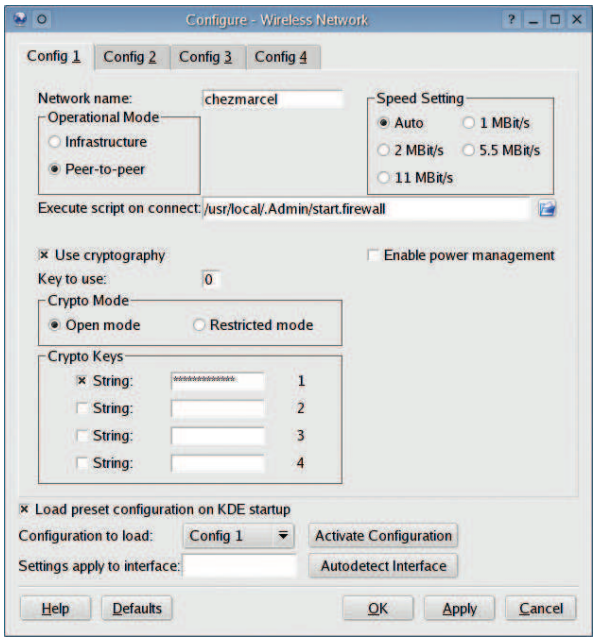
A ***KWiFiManager*** alapértelmezésként nem települ. Ha ellenőrizni akarjuk, hogy rendszerünkön elérhető-e, vessünk egy pillantást a KDE menüre, vagy próbáljuk meg kiadni a `kwifimanager` parancsot.

Induláskor a ***KWiFiManager*** megjeleníti a jelerősséget, a kapcsolat sebességét, a hozzáférési pont leírását, a helyi IP címünket, valamint a kommunikációs csatorna frekvenciáját. A kapcsolat minőségétől függően a ***TOP*** (csúcs), ***EXCELLENT*** (kiváló) vagy az ***ULTIMATE*** (kb. félelmetes) szavak egyikét láthatjuk a numerikus kijelző mellett. Ezzel együtt a tálcán is megjelenik egy kis ikon, amin szintén látható a megfelelő numerikus érték és annak grafikus megfelelője. Ha a kapcsolat minőségének időbeli alakulását akarjuk megjeleníteni, válasszuk a ***File*** menüből a ***Connection statistics*** (***Kapcsolati statisztika***) pontot (*3. ábra*).

Mindazok számára, akik becsapva érzik magukat, mert hiányolják a tudományos-fantasztikus filmekben a számítógépek által keltett apró zajokat, csipogásokat és egyéb prüntyögéseket, a ***KWiFiManager***-nek van hangszolgáltatása is. Bekapcsolhatunk egy csipogó hangot, ami a rádiós kapcsolat minőségétől függően magasabb vagy alacsonyabb tónusban „pittyeg”. Ehhez válasszuk a ***Config*** menüből az ***Acoustic Scanning*** pontot. Mindenkit szeretnék figyelmeztetni, hogy a csipogás rövid távon ugyan mulatságos, hosszan hallgatni azonban nem valami jó szórakozás.

A ***KWiFiManage*** messze több egy térerőkijelzőnél. Lehetőséget ad például arra, hogy automatikusan kapcsolódjunk különböző hálózatokhoz. Ezek beállításához válasszuk a ***Config (Beállítások)*** menüből a ***Configuration Editor (Beállításszerkesztő)*** pontot. A program előbb bekéri a root jelszavát, majd megjelenik egy valamivel nagyobb ablak. Ennek a tetején négy fület látunk, amelyeken a Config1...4 feliratok olvashatók. Mindegyik fülhöz ugyanolyan ablak tartozik, amelyekben megadhatjuk az egyes hálózatok különböző paramétereit. Ezekről mindjárt bővebben is szólok, de most azt szeretném, ha vetnénk egy rövid pillantást a ***KWiFiManager*** ablakának alsó részére. Ott virít egy jelölőmező, amivel azt írhatjuk elő, hogy az itt megadott beállításokat a KDE indulásakor automatikusan jussanak érvényre. Ha például az időnk túlnyomó többségét a saját irodánkban töltjük, akkor célszerű ennek a helynek a hálózati beállításait megtenni alapértelmezettnek.

Akkor most térjünk vissza a beállítási lehetőségekhez. A legalapvetőbb információ, vagyis az adott hálózat neve rögtön ott van az ablak tetején. Természetesen minden egyes beállításhoz tartozik egy ilyen. Ha azt akarjuk elérni, hogy induláskor a KDE egyszerűen keresse meg az éppen elérhető hálózati kapcsolatot, és használja az ahhoz tartozó beállításokat, akkor az `ANY` nevet alkalmazzuk. A névtől jobba látható a kapcsolat sebességének megadására szolgáló



4. ábra Akár 4 kapcsolatot is kezelhetünk

mező. Én ezt mindig az `Auto` beállításon hagyom. Közvetlenül a beállítások alatt találunk egy mezőt, amiben annak a szkriptnek a nevét adhatjuk meg, amelynek csatlakozáskor le kell futnia. Ez természetesen akármi lehet, héjprogram is. Vannak, akik ide megírják a saját, különleges igényeikhez igazodó szkriptet, de olyanok is akadnak, akik elfogadják a rendszer által ajánlott alapbeállításokat. Jómagam bizonyos a tűzfalat beállító programokat szoktam itt lefuttatni attól függően, hogy milyen szolgáltatásokat akarok elérhetővé tenni, illetve hogy éppen hol vagyok. Ha már a biztonságnál tartunk van kissé lejjebb egy jelölőmező, amivel a kapcsolat titkosítását írhatjuk elő. Ha olyan hozzáférési ponthoz csatlakozunk, amely WEP titkosítást használ, erre mindenképpen szükségünk lesz. Az alkalmazás ablakban néhány további beállítási lehetőséget találunk. Megadhatjuk például a használni kívánt jelszavakat. Ha mindezzel végeztünk, akkor kattintsunk az ***OK*** vagy az ***Alkalmaz*** gombra, (Utóbbi nyilván akkor hasznos, ha további beállításokat is meg szeretnénk adni.) És ezzel meg is volnánk. Talán érdemes még megjegyezni, hogy ugyanehhez a beállítóablakhoz a ***KDE Vezérlőközpontból (KDE Control Center)*** is hozzáférhetünk ***kcontrol*** néven. a megfelelő pontot az Internet beállítások között kell keresnünk.

Nos kedves barátaim, ismét eljött a búcsú perce, de ahogy tudjátok, ***Marcel*** soha nem veszi túl komolyan a zárórát. Igyatok hát még egy pohárral e jó borból, és élvezzétek a vezeték nélküli hálózat áldásait.

Linux Journal 2004. szeptember, 125. 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.

© Kiskapu Kft. Minden jog fenntartva

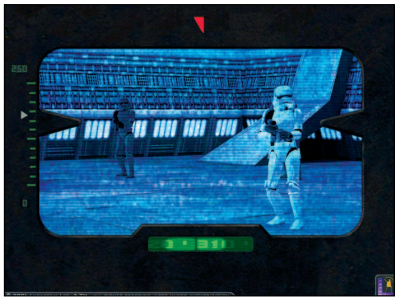
© Kiskapu Kft. Minden jog fenntartva

Jedi Knight2: Jedi Outcast!

A PC monitorán valahogy sosem volt képes engem megragadni a Star Wars univerzum. A filmek nagyon tetszettek, de ez volt minden. Egészen addig, amíg egyszer véletlenül, a kezembe nem akadt a Jedi Knight második része, amelynek varázslatos világával most végre a Linux felhasználók is megismerkedhetnek.

Szinte tökéletes az illúzió, minden pillanatban a Star Wars galaxis világában érezhetjük magunkat. A történet szerint mi Kyle Katarn-t alakítjuk, aki az új köztársaságban röpköd hajójával, no és bájos hölgy segítőjével, aki mellesleg igen közel áll a szívéhez. Ekkor kapjuk ***Mon Montha*** szenátor asszonytól a megbízatást, (egy animáció keretében) hogy nézzünk szét egy bolygón, amely egy régi birodalmi állomásnak ad otthont, és mellesleg igen gyanús. Természetesen elvállaljuk, melynek során végérvényesen belegabalyodunk egy szövevényes összeesküvésekkel, és sok-sok legyilkolandó ellenséggel tarkított történetbe. Ekkor sajnos még semmit sem tudunk a Jedi tehetségünkről, olyannyira, hogy egy csirke lézerpuskával vagyunk kénytelenek irtani az ellent. Quake rajongók előnyben.

Legalábbis az elején. A játék ugyanis az első küldetésig – híven a Jedi Knight sorozathoz – FPS módban játszható, ám a Jedi tudományok ismeretével a hátunk mögött már a jó öreg ***Lara Croft*** nézetben folytatjuk a kalandot. Természetesen opcionálisan ekkor is választhatjuk az FPS módot, de tapasztalataim szerint határozottan kényelmetlen. Az első küldetés befejezése után (jó szereplésünk okán) teljesen ráállítanak minket az ügyre. Küldetésünk az, hogy kiderítsük, mi is történik itt. Ekkor átküldenek a Yavin IV-re, ahol bizony maga a nagy ***Luke Skywalker*** kezdi meg képzésünket a Jedi Akadémián. Érdemes nagyon



figyelni, (és a lehető legjobban elsajátítani a billentyűzet használatát) mert az Erő használata nélkül gyakorlatilag esélytelenek vagyunk. A billentyűzet és egér kombinálásának ismerete pedig garancia arra, hogy az – egyébként elég kemény – Dark Jedi harcosokat is át tudjuk küldeni az örök vadászmezőkre.

A pályák változatossága igazi élmény, ötletességük pedig példaértékű. Küldetést fogunk teljesíteni a Yavin IV-en, A Bespin-i felhővárosban, sőt kalózoktól, bűnözőktől hemzsegő űrkikötőben, kocsmában, birodalmi csillagrombolón. Harcrendszere igen érdekesre sikeredett, lévén a sötét harcossal folytatott küzdelem animált végkifejlettel rendelkezik. A közkatonák gyilkolása teljesen hétköznapi. A tápos ellenfelekkel való csatározások végén a játék a saját motorjával renderelt lassított animációt vetít, az egyik küzdő fél összeeséséről. Ez jobb esetben az ellenfél, rosszabb esetben Kyle. Érdekesek, izgalmasak, sőt határozottan felemelőek ezek a fénykard csaták, olyannyira, hogy már vágyik az ember egy jó kis csihipuhira. Fantasztikusra sikeredett ez a küzdelem, a legkülönlegesebb pedig az, amikor két sötét lovag támad ránk egyszerre. Figyelem, a sötét harcosok nagyon gyorsak. És nagyon veszélyesek...

Külön kiemelendő, hogy a fejlesztők nagyon odafigyeltek a fokozatosságra. Valamelyest a szerepjátékokra emlékeztet, hogy a Jedi tudományokban való elmélyülés új mozdulatokat, vagy az Erő használatát eredményezi. Ráadásul bizonyos időközönként leírást találunk egy új Jedi trükkről, mintha új fegyvert vennénk fel. (Ez viszont klasszikus FPS elem.) Így a játék közepe felé már képesek leszünk úgy megfojtani egy katonát mint maga ***Darth Vader*** (sötét düh) vagy olyan villámeffekteket produkálni, mint amilyeneket az uralkodó művelt a Jedi visszatér című film végén. Mindezek mellett a fegyverarzenál is igen változatos, a ***Chubakka***-féle „nyílpuska" ugyanúgy megtalálható, mint a sima lázadó lézer fegyver, vagy a távcsöves orgyilkos műszerek. Közepes fokozaton a játék közepén szinte már legyőzhetetlenek vagyunk. Pusztulásunkat kizárólag a figyelmetlenség, vagy meg-

gondolatlanság okozhatja. Utóbbira nagyon kell vigyázni, lévén akadnak csapdák, amelyeket esetleg az ellenfelek állítottak nekünk. Ha azonban hideg fejjel, megfontoltan lépkedünk az ellenséges területen, aligha kerülhetünk izzasztó helyzetekbe. Félelmetes látvány, amint egy szakasz katona közé beugorva fénykardal rendet vágunk közöttük.

És amitől tényleg a játékban érzed magadat...

A számos ismert szereplő felbukkanása sokat dob az illúzión, hisz könnyebben mélyedünk el a történetben. A jól ismert Star Wars zenék, és a filmhez tökéletesen hű hangeffektusok végleg elvarázsolnak. Gyakorlatilag pont azt hallja az ember, amire a filmre gondolva számítana. Ugyanez a lények, és a „játék biológiájára is igaz", gyakorlatilag „új szörny" nincsen. Mindet láttuk már a filmekben, még ha csak villanásnyira is. És ha ki akarod próbálni, milyen vezetni egy birodalmi lépegetőt, akkor úgysem fogod kihagyni ezt a játékot...

A technológia

A játék lelke, a már kicsit réginek számító Quake 3 Team Arena motor, amely némi kozmetikán esett át, és remek látványt nyújt. Sajnos ennek az a következménye, hogy míg a Quake3 jól futott egy 500 MHz-es Celeron-on egy voodoo 3 szintű kártyával, addig ennek a játéknak ez a konfiguráció egy kicsit karcsú. Az első generációs GeForce, illetve a 6-700 MHz-es Pentium 3 itt bizony kötelező. Azt már talán mondanom sem kell, hogy a játék 256 MB memória alatt nem érzi jól magát. További érdekesség, hogy a program adatszervezése is azonos a Quake3-éval, így egy közönséges zip tömörítővel percek alatt meg lehet „műteni". Jó tudni, hogy a játék zenéi valójában *.pak* fájlokon belüli mp3 állományok, így gond nélkül lecserélhetők. (Persze ügyelni kell a bit rate helyes beállítására.)

Telepítés

Talán ez az a rész, amely mindenkit a leginkább érdekel. A játéknak (mint oly sok társának) sajnos nincs linuxos binárisa. Létezik viszont egy projekt, melynek célja, hogy *winex* kompatibilis telepítőket, és binárisokat hozzanak létre, kifejezetten az adott játékhoz. Nincs tehát más dolgunk, mint a Jedi Knight második részének a csomagját letölteni a ➲ http://liflg.sourceforge.net/ weboldalról (körülbelül 7 MB). Ez egy .run kiterjesztésű fájl, amely rendszergazdai módban gond nélkül futtatható (csak így tud írni a megfelelő könyvtárakba).
A letöltött telepítőhöz szükséges még a játék CD lemeze is (a Windows alatt használatos lemezről van szó), és természetesen a *winex* legfrissebb változata. A leírás szerint a játék *winex 3.x* alatt már futni fog. Talán már nem is kell említenem, hogy a telepítő a Loki telepítőrendszerét használja, így nem alkalmas a fájlrendszerek önműködő befűzésével együttműködni. (Ezt a szolgáltatást tehát ki kell kapcsolni, de telepítés után visszaállítható, hiszen a CD-re többet nem lesz szükségünk.) Nincs más hátra, mint belevetni magunkat a Star Wars hangulatos világába.

Dancsok „strogg" Zoltán
(strogg@mail.tvnet.hu)
Jelenleg technikai szerkesztőként dolgozik a BME-OMIKK-nál, ahol oktat is. Emellett egyetemi képzésben vesz részt, programozó matematikus szakon. Négy éve foglalkozik Linuxszal. Szabadidejében operációs rendszereket gyűjt és weblapot vezet.

© Kiskapu Kft. Minden jog fenntartva

© Kiskapu Kft. Minden jog fenntartva

Szoftverjog – A szerzői jogi alapfogalmak

Miután az előző részben valamelyest tisztáztuk a szoftver fogalmát, ideje megismerkednünk néhány egyéb, a szerzői jog szempontjából szintén nélkülözhetetlen fogalommal.

Szerző az, aki a művet megalkotta. Ez eddig nem túl meglepő. Ugyanakkor, ha belegondolkodunk abba, hogy jogi személyek milyen sok esetben hiszik magukat szerzőnek, láthatjuk, hogy a helyzet korántsem zavarmentes.

A szerzői alkotás képessége kizárólag magánszemélyeknek (emberek) adatott meg, hiszen a törvény szerint a szerzői jogi védelem az alkotást „a szerző szellemi tevékenységéből fakadó egyéni, eredeti jelleg alapján illeti meg". A természetes személyeken kívül ugyanakkor vannak jogi személyek (Kft., Rt) illetve jogi személyiséggel nem rendelkező társaságok (ilyen a Bt, és a Kkt) is, amelyekkel szintén foglalkoznunk kell.

Aztán itt van még a mesterséges intelligencia problémája. Ha egyszer eljön Dougles Adams kora, és lesznek emberi öntudattal bíró, szellemes robotjaink is (Marvin) akkor talán az említett paragrafust is kiterjesztően kell majd értelmeznünk. Egyelőre azonban jobb, ha beletörődünk, hogy a hatályos magyar szabályok szerint sem Kft, sem egy számítógépes szoftver nem tekinthető szerzőnek. Utóbbi ugyan képes minimális segítséggel más szoftvereket előállítani, de ez azért még nem az előbb említett „robotszellemesség". A külföldi irodalomban ugyanakkor olvashatunk kísérleteket arra, hogy ezt a kicsit visszás helyzetet tisztázzák, például az úgynevezett „közvetett szerzőség" lehetőségével. Itt elsősorban az intelligens fejlesztési környezetekre kell gondolni, ahol vitatott, hogy a környezetet használó programozó pár kattintása valóban szerzői teljesítmény-e, hiszen a fejlesztés jelentősebb részét sokkal inkább maga a program adja. Érdekes, sőt mulatságos ellentmondás, hogy ugyanilyen alapon alkotótevékenységnek tekinthetjük azt, amit egyes a saját kódjukat újraformálni képes férgek művelnek...

Amerikai mintákat látva azonban azt tapasztalhatjuk, hogy ott szinte csak jogi személyek nevezik meg magukat szerzőként. Ez az ott alkalmazott copyright rendszer egyik sajátossága. Ez a rendszer ugyanis a szerzőséget „másban méri". Egy film esetében például náluk a producer lesz szerzőként feltüntetve, ő lesz a JOGOSULT (csupa nagybetűvel), mivel ő teremtette meg a finanszírozási feltételeket, ő választotta ki az alkotókat, s nem utolsó sorban ő „vásárolta ki" jogaiból az összes többi alkotót. A copyright rendszer egyik lételeme, hogy minden jog eladó. A kontinentális szerzői jog ezt nem teszi lehetővé. Nálunk jogosultságokat csak felhasználási szerződés keretében (de ott akár kizárólagos jelleggel is) lehet átengedni.

Mindez persze messze nem jelenti azt, hogy a magyar cégeknek ne lennének az alkotásokhoz fűződő jogaik. A művel való rendelkezés jogát, valamint az igazán pénzt érő jogokat (vagyoni értékű jogok) megszerezhetik pl. egy kizárólagos felhasználási szerződés megkötésével vagy törvényi felhatalmazásra a vagyoni jogokat azért, mert a programozó velük áll munkaviszonyba. Erről azonban bővebb részleteket majd a vagyoni jogokról szóló cikkben találhatunk…

Ráadásul a polgári törvénykönyv rendelkezései szerint is megilletheti a nem természetes személyeket (az élő emberen kívül minden más jogalany) személyhez fűződő jogok védelme, kivéve, ha ez a védelem jellegénél fogva csak magánszemélyekre értelmezhető.

Lássunk egy példát! Ha megharagszunk egy Kft-re, és hangos szitkok közepette a Deák téren kora délután elkezdjük róla terjeszteni, hogy az anyukája örömlányként keresi a mindennapi betevőt, akkor nevezett személyhez fűződő jogait nem sérthetjük meg, neki csak alapítói vannak, anyukája nincs(Így a személyhez fűződő védelem értelmezhetetlen). Ha viszont ugyanezt a cég ügyvezetőjéről állítjuk, aki magánszemély, akkor jó eséllyel becsületsértést követünk el, amelynek már jól meghatározott törvényi következményei vannak.

De ha egy cég jó hírnevét csorbítjuk az által, hogy azt terjesztjük róla: használhatatlan szoftvereket gyárt (amennyiben nem tudjuk bizonyítani) szintén jogsértést követünk el. Hiszen „A jó hírnév sérelmét jelenti különösen, ha valaki más személyre vonatkozó, azt sértő, valótlan tényt állít, híresztel, vagy való tényt hamis színben tüntet fel." Ptk. 78.§ (2)

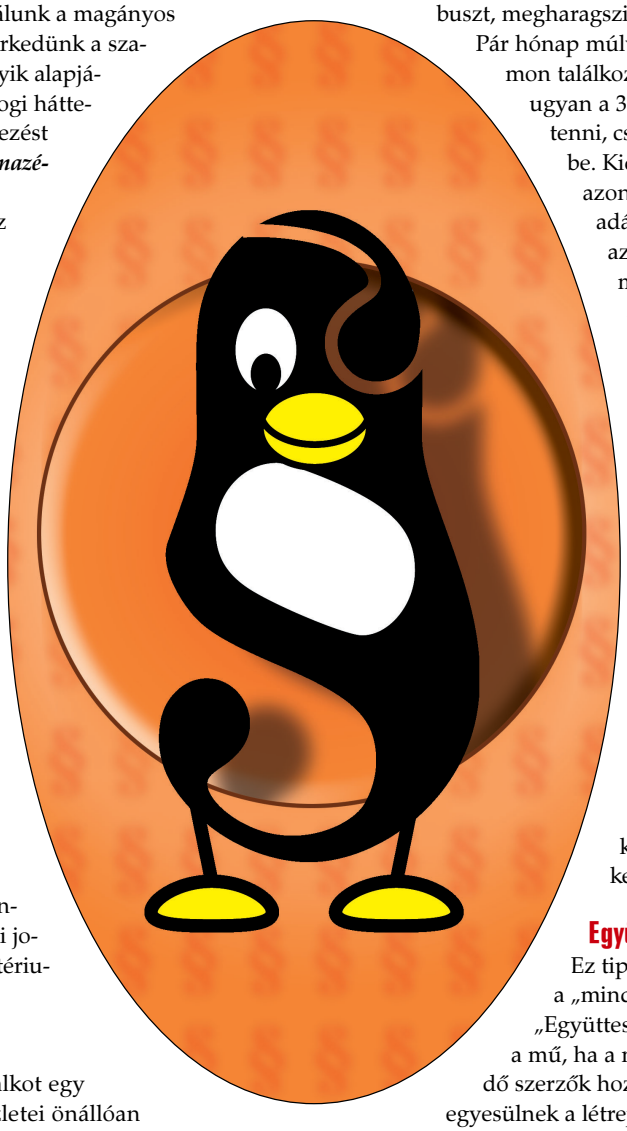
Korhatár

Ha a 6 éves unokaöcsénk hosszú éjszakákon át egy igen kiváló játékprogramot fejleszt, joggal gondolhatunk arra: „Egek, ő még olyan kicsi, hogy biciklit sem vehet magának, mi lesz akkor a programjával, és a szerzői jogaival?" Mindenki megnyugodhat, a szerzőség nem korfüggő. Öcsi ugyanúgy lehet szerző, mint mi magunk vagy éppen a 103 éves nagymamánk.

Azt azonban tudnunk kell, hogy jogai védelmében – ha mondjuk a szomszéd Karesz, aki egy 25 éves zsivány, el-

orozná a programot és saját neve alatt beküldené egy pályázatra – törvényes képviselője (anyuka, apuka) tehet jognyilatkozatot. (Ilyen például, ha feljelenti a szomszédot.) Nem jogi jellegű nyilatkozatként természetesen mi magunk is megdorgálhatjuk Károlyt.

Ezen a ponton egy időre elválunk a magányos programozóktól, és megismerkedünk a szabadszoftveres fejlesztések egyik alapjának, a „csapatmunkának" a jogi hátterével. Először is négy új kifejezést kell bevezetünk. Ezek a ***származékos művek***, a ***közös művek***, az ***összekapcsolt művek*** illetve az ***együttesen létrehozott művek***.



Származékos művek

Némi fantáziával rögtön érezhető, hogy a származékos művek esetében létezik egy alapul szolgáló (eredeti) és egy abból valamilyen úton-módon (kémiai folyamatok többnyire kizárva) származtatott másik alkotás. A származékos művek alapvető fajtái az átdolgozás, a feldolgozás és a fordítás. Ezekkel egy későbbi számban majd részletesen foglalkozunk, egyenlőre azonban legyen elég annyi, hogy ezekben az esetekben a két alkotás időben követi egymást, valamint az alapul szolgáló mű (szoftver) már önmagában eleget tesz a szerzői jogi védelemhez szükséges kritériumoknak.

Közös mű

Ha több szerző együttesen alkot egy olyan művet, amelynek részletei önállóan nem használhatóak fel, akkor a szerzői jog együttesen, kétség esetén pedig egyenlő mértékben illeti meg a szerzőtársakat. Tipikusan ilyen helyzet az, amikor egy magyar és egy átmenetileg Japánban élő magyar fejlesztő az „Akciófront a pontos 182-es buszokért" internetes fórumon véletlenül találkozik. Mivel mindkettőnek régi vágya, hogy Budapest szövevényes közlekedési rendszerének ezen a „viszonylatán" hathatós segítséget nyújtsanak a BKV-nak, megegyeznek a fejlesztésben. És mivel a probléma megoldása tényleg nem tűr halasztást, felváltva dolgoznak rajta. Az időeltolódás okán a Magyarországon dolgozó szerző nappal ír, aztán esteledve a japán kolléga, aki eddig aludt gyorsan meghallgatja a fejleményeket, és folytatja a munkát.

Az elkészült programot nyilvánvalóan nem lehet kódsoronként szétdarabolni. Alkotói szerzőtársak, alkotásuk pedig közös mű.

Összekapcsolt művek

Maradjunk az előző példánál, de módosítsuk egy kicsit a feltételeket. A magyar fejlesztő elkezd egy közlekedésrendező programot írni, félig elkészül vele, aztán mivel lekési az életben egyszer tényleg pontosan érkező 182-es buszt, megharagszik, és felhagy a fejlesztéssel. Pár hónap múlva azonban az említett fórumon találkozik egy másik fejlesztővel, aki ugyan a 30A villamost szeretné pontossá tenni, csak véletlenül épp ide tévedt be. Kiderül, hogy egész véletlenül azonos nyelven programoztak, ráadásul egyikük alkotásából pont az a rész hiányzik, amit a másik már megírt. Összeillesztik a részleteket, amelyek már magukban is működtek (tehát pl.: az egyik már hibátlanul kezelt a 30A villamos hétvégi és munkanapokra vonatkoztatott indulási időpontjait, a másik pedig koordinálta a nappali és éjszakai járatsűrűséget), és segítségükkel pontossá teszik mind a 182-es buszt, mind a 30A villamost. Jelen esetben társszerzői minőségükben tűnnek fel, és amikor a BKV megvásárolja a szoftvert, és szeretné annak egyes részeit hozzáilleszteni a hipermodern közlekedési szoftveréhez, mindkét szerző beleegyezését meg kell szereznie.

Együttesen létrehozott művek

Ez tipikusan az az eset, amikor a „mindenki egyért" elv érvényesül. „Együttesen létrehozottnak minősül a mű, ha a megalkotásában együttműködő szerzők hozzájárulásai olyan módon egyesülnek a létrejövő egységes műben, hogy nem lehetséges az egyes szerzők jogait külön-külön meghatározni." (Szjt. 6.§) A tipikusan ilyennek tekinthető a vállalati szoftverfejlesztés, illetve egyes multimédiás, audiovizuális művek. Ebben az esetben a szerzők jogutódjaként azt a természetes vagy jogi személyt, illetve jogi személyiséggel nem rendelkező gazdasági társaságot illeti meg a szerzői jog, amelynek kezdeményezésére és irányításával a művet létrehozták, és amely azt a saját nevében nyilvánosságra hozta. Hangsúlyozni kell, hogy a társaság itt sem válik szerzővé, csak bizonyos jogok jogosultjává.

Dr. Dudás Ágnes (dudas.agnes@abend.hu) ügyvédjelölt, az FSF egyik aktivistája. 2004-ben végzett az ELTE Jogtudományi Karán. Szakdolgozatát a szoftverek szerzői jogi védelméről írta, a 2003-as évet pedig e terület kutatásával a berlini Humboldt Egyetemen töltötte.

© Kiskapu Kft. Minden jog fenntartva