Beköszöntő

Közeleg az év vége, és bár legtöbbünk életében a falinaptár lecserélése tulajdonképpen semmiféle tényleges változást nem jelent, mégis bevett szokás, hogy ilyenkor visszatekintünk az elmúlt 12 hónapra, és megpróbáljuk megtalálni a hangsúlyos pontokat.

A ***Linux*** esetében az „év témájának" talán a szoftverszabadalmak – továbbra is aktuális – ügyét tekinthetjük.

A Linuxvilág alapvetően szakmai lap, így cikkeink túlnyomó többsége az érdekes műszaki lehetőségekkel, újításokkal, valamint a kezdő felhasználók segítésével foglalkozik. Egy ilyen méretű „vörös posztó" mellett azonban mi sem mehettünk el szó nélkül. Éppen ezért indítottuk útjára néhány hónappal ezelőtt ***Dr. Dudás Ágnes*** sorozatát, amely közérthető bevezetést ad a témába, s így reményeink szerint elősegíti azt, hogy a szabad szoftverek alkotói és használói átlássák, pontosan milyen következményei is lehetnek annak, ha szabadalommal védenek például olyan műszaki megoldásokat, amelyeket bármely értelmes, konstruktív ember fél óra alatt képes kigondolni a sarokban üldögélve.

Bízunk benne, hogy a tőkés társaságok egyre több területen érzékelhető és egyre nyilvánvalóbb világuralma nem terjed majd ki a szabad szoftverek világára is, holmi gazdasági érdekek miatt nem köti gúzsba a műszaki fejlődést, és nem kényszerít intellektuális illegalitásba eddig szabadon alkotó, és a köz érdekeit szolgáló embereket.

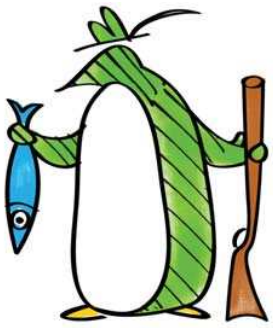
Bár a többi témát sem hanyagoltuk el, és a szokásos sorozatok is folytatódnak, ez évi utolsó számunk két leghangsúlyosabb témája a biztonságtechnika és a ***PHP*** nyelv. Az előbbi témát három cikk is érinti. Szó esik a ***SELinux***-ról, ***Mick Bauer*** folytatja a ***Linux*** fájlrendszerek biztonsági kérdéseinek tárgyalását, ***Hasnain Antique*** pedig azt mutatja be, hogyan igazodhatunk el könnyen abban az adattengerben, amely egy komolyabb hálózat biztonsági átvizsgálása során keletkezik.

A ***PHP 5*** nyelv objektumközpontú lehetőségeinek használatáról ***Komáromi Zoltán*** kezdett el egy négy részesre tervezett sorozatot, ***Heilig (Cece) Szabolcs*** pedig a nyelv különleges lehetőségeiről ír majd néhány cikket.

És most lássuk a magyar szerzők sorozatait! ***Fábián Zoltántól*** megtudhatjuk, hogyan tervezhetünk cégünk számára reklámsávot és logót a ***GIMP 2.0*** segítségével, ***Auth Gábor*** folytatja a ***FreeBSD*** bemutatását, ***Illés Viktor*** pedig a ***Linux*** és a mobil eszközök összeházasításának nehézségeit taglalja. Ebben a részben a grafikus felület beállításáról lesz szó. ***Garzó András*** a hálózatok lelkivilágát tárgyló sorozatának immár a 12. részénél tart, míg ***Beszédes Balázs*** a felhasználók lelkivilágának és viselkedésének vizsgálati módszereit mutatja immár műszaki szempontból.

Kellemes időtöltést, jó szórakozást kíván a Linuxvilág magazin csapata!

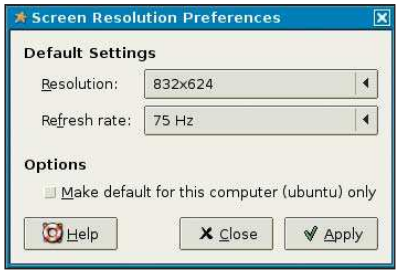
Programvadászat



Decemberi korongunk legnagyobb részét az ***Ubuntu Linux Live CD*** foglalja el. Ez egy ***Debian/GNU Linux*** alapú terjesztés. Mostanában szerencsére egyre többször bukkannak fel ilyen kiadások (ilyen például a ***KNOPPIX***). A ***Debian*** előnyeit és hátrányait mérlegelve a fejlesztők régóta érdemesnek találják azt arra, hogy a saját munkáik alapjául válasszák. Ez a sor a ***Storm Linux***-szal kezdődött. Sajnos a ***Stormix Technologies Inc.*** körülbelül egy év után feladta a fejlesztést. A következő nevező a ***Corel Linux*** volt. Mint a névből sejthető itt a ***Corel*** grafikai programokat gyártó cég állt a háttérben azzal a céllal, hogy saját operációs rendszert hozzon létre programjaihoz. Nagyon jó eséllyel indultak, de végül „lebeszélték" őket. Ezek után a ***Xandros*** vette a kezébe a ***Corel*** által elfelejteni akart terjesztést, amely a mai napig is létező kereskedelmi változat. Aztán a ***Progeny Linux*** következett amely szintén nagyon ígéretesnek indult, de nem jött be a kereskedelmi számításuk. Ők is feladták, és inkább a tanácsadásra koncentrálnak. A ***Linux*** népszerűsítésében nagy szerepet kapott többek között a mi korongunkon is megjelent ***KNOPPIX Linux*** szintén ***Debian*** alapokon nyugszik, és ilyen az ***Ubuntu*** is. Az ok ami miatt a fejlesztőknek érdemes a ***Debiant*** alapnak választani a hatalmas programarzenál, ami azonnal elérhető ***apt*** forrásban. A fenti felsorolás természetesen nem teljes, inkább csak rövid áttekintés.

Az Ubuntu

Az ***Ubuntu*** egy ősi afrikai szó jelentése: „vagyok aki vagyok, mert mindenki az aki". A fejlesztők szigorúan veszik azt a célt, hogy az ***Ubuntu*** mindig mindenki számára szabadon elérhető legyen. Nagy hangsúlyt fektetnek a nemzetköziségre, a programok és



dokumentációk fordítására. Céljuk, hogy ***„minél több emberhez juthasson el az Ubuntu"***. Hat hónapos kiadási ciklussal számolnak, és a megjelenő rendszert 18 hónapig támogatják, biztonsági frissítésekkel és hibajavításokkal. Ez hatalmas előny a ***Debiannal*** szemben.

Felhasználás

Az ***Ubuntut*** használhatjuk asztali gépeken vagy kiszolgálóként is. A jelenlegi kiadás az ***Intel x86***, ***AMD64*** és ***Power PC*** architektúrákat támogatja. Több mint 1000 programot használhatunk azonnal. Van benne 2.6-os rendszermag, a 2.8-as ***GNOME***, és természetesen minden feladathoz találunk megfelelő programokat, legyen szó akár böngészésről, irodai feladatok ellátásáról, webkiszolgálóról vagy játékokról.

Érdekességek

Az Ubuntu két változatban létezik. Van ***Live CD*** és ***telepítő CD***, amelyek felépítésében természetesen vannak különbségek, Mindkettőt telepíthetjük, viszont a hardverfelismerés és az ***X*** felbontásának kitalálása másként történik. Számomra nagyon érdekes és kifejezzen jó ötlet, hogy a ***root*** felhasználó alapból tiltva van. A telepítés során létrehozott első felhasználó kap ***sudo*** jogokat, vagyis a ***root*** felhasználó jogaival futtathat programokat a ***root*** jelszavának ismerete nélkül, ez biztonsági szempontból előnyös megoldás. Következő számunkban bővebben írunk erről a terjesztésről.

Korongunkon helyet kapott a szokásos ***Bitdefender*** frissítés is. Néhány a mostani cikkekhez tartozó csomag és forráskód az előző, tehát a 65. CD-n kapott helyet, mivel az ***Ubuntu Linux*** miatt most nem fértek volna fel. Ezeknél a cikkeknél tehát nem elírás a 65. CD/Magazin kezdetű sor. Aki esetleg nem rendelkezik a Linuxvilág előző számával az a cikkekhez tartozó anyagokat megtalálhatja a ➲ ***melleklet.linuxvilag.hu*** címen is.

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

Megnyíló Ingres

A *Computer Associates* bejelentette, hogy *Ingres r3* adatbázisát mind *Linux*, mind *Windows* operációs rendszerre

Ingres®

általánosan elérhetővé tette. A cég még augusztusban tette közzé az adatbázis-kezelő megnyitott forráskódját, amelynek további operációs rendszerekhez – 32 és 64 bites *Sun Solaris*, *HP/UX* és *IBM AIX*, továbbá *HP Tru64* és *OpenVMS* – készített változatai idén és jövőre várhatók. Az *Ingrest* nagyjából egy évtizede az *Ask Computer Systemstől* vásárolta meg a *CA*, ám a jó nevű rendszer piaci részesedése az utóbbi időben folyamatosan csökkent. A nyílttá tétellel a felhasználói tábor újbóli növekedését várják, nem is alaptalanul, hiszen augusztus óta több mint 15 ezer letöltést jegyeztek fel.

➲ opensource.ca.com/projects/ingres

Pálfordulás

Az *Intel* részéről is elismerték, a megahertzek kora véget ért: a cég törölte termékfejlesztési úti tervéből a 4 GHz órajelű *Pentium 4* processzort. Az *Intel* hosszú ideig tartotta magát ahhoz a nézethez, hogy az órajel növelésével megfelelően fokozható a teljesítmény is – bár az új processzorszámozás bevezetése arra utal, hogy talán maguk sem hittek az egészben. A legújabb fejlemények szerint az *Intel* legnagyobb órajelű egymagos processzora a 3,8 GHz-es lesz. Az órajel növelésének elsősorban a hatalmas energiafogyasztás és hőtermelés állja útját, így a teljesítmény fokozását a továbbiakban más módszerekkel kénytelenek megoldani. Ilyen például a gyorsítótár méretének növelése, ami ráadásul viszonylag könnyen kivitelezhető; a sokat emlegetett kétmagos processzorok bevezetése valamint a 64 bitre való áttérés, de az *Intel* részéről a virtualizációs és biztonsági szolgáltatások bevezetését, bővítését is számba veszik, mint a „felhasználói élmény” fokozásának lehetséges módját. Tétlenkedésre ugyanakkor nincs idő, a versenytársak ugyanis már rendelkeznek 64 bites és kétmagos processzorokkal is, az *Intelnek* rövidebb távon is újdonságokkal kell előhozakodnia, ha meg akarja tartani vásárlóit.

Xen 2.0

Elkészült a nagyteljesítményű virtuálisgép-alaprendszer, a *Xen 2.0*-s változata. A *Xen* fejlesztésének célja az, hogy egyetlen *x86* alapú számítógépen közel natív sebességgel több operációs rendszert is lehessen futtatni: a vendég operációs rendszerek átültetett változataival dolgozva a teljes értékű számítógépet emuláló megoldásoknál sokkal jobb teljesítményt képes biztosítani, sőt, a felhasználók által észlelt teljesítménycsökkenés gyakorlatilag elhanyagolható ahhoz képest, mintha a számítógép natívan futtatná a vendég rendszert. A *Xen* használatához a futtatni

Xen

kívánt rendszermagokat ugyan módosítani kell, ám a változások a programkönyvtárakra és az alkalmazásokra már nem terjednek ki. A *Xen* jelenleg a *2.6.9*-es és a *2.4.27*-es *Linuxot* és a *NetBSD*-t támogatja, de a közeljövőben további rendszerek támogatását is megvalósítják. A *Xen* a *Cambridge*-i *Egyetem* nagy távolságú, elosztott adatfeldolgozó infrastruktúrák kiépítését célzó *Xenoservers* tervezetének egyik eleme, a jövőben olyan ígéretes szolgáltatásokkal tervezik bővíteni, mint például a fürtkezelés, a processzorok közötti terheléselosztás és a többprocesszoros vendég rendszerek futtatása.

➲ www.cl.cam.ac.uk/Research/SRG/netos/xen

A bástyák ledőlnek

A felröppent hírek szerint a *Sharp* hamarosan kivonja az amerikai piacról linuxos *Zaurus* zsebtitkár modelljeit, így ezek kizárólag *Japánban* lesznek megvásárolhatók. A felmérések szerint a mobiltelefon nélküli zsebtitkárok értékesítése a közeljövőben komoly csökkenés elé néz – sőt, a folyamat már be is indult –, az ilyen készülékek helyét egyre inkább az intelligens mobiltelefonok és a mindent egyben jellegű tenyérgépek veszik át. A kivonás mellett további érv lehet, hogy a linuxos zsebtitkárok a szűkebb rajongói táboron túl nem nagyon nyerték el a vásárlók tetszéseit, a piacot továbbra is a windowsos és a *Palm OS* alapú termékek uralják. Nemrég a *Sony* és a *Toshiba* is hasonló döntést hozott.

Mobil… mi is?

A *Nokia* bemutatta *7710*-es típusú, a hagyományos mobiltelefonokra immár a legcsekélyebb mértékben sem

hasonlító telefonját. A szélesvásznú érintőképernyővel ellátott készülék kiválóan példázza a mobiltelefonok és a zsebtitkárok összeolvadását. Kezelése utóbbiakhoz hasonlóan egy tollal történik, képességei pedig mindkét készülékcsoportot idézik: internetes böngésző, zene- és mozgóképlejátszó, 1 MP felbontású kamera, *FM* rádió, *MP3* lejátszó, szövegszerkesztő, táblázatkezelő, személyi adatkezelő szolgáltatások, kézírás-felismerés, képernyőn megjelenő billentyűzet. Természetesen a legújabb és a jól bevált szolgáltatások támogatása sem hiányozhat belőle, az internetre *EGPRS* vagy *HSCSD* összeköttetésen keresztül csatlakozik, leveleinket pedig *POP3*, *SMTP* és *IMAP* protokollon keresztül egyaránt képes elérni. Alapmemóriájának mérete 90 MB, amit legfeljebb 512 MB méretű *MMC* kártyával tudunk bővíteni. A 7710 Ázsiában már idén kapható lesz, míg Európában inkább csak húsvétra lephetjük meg magunkat vele.

➲ www.nokia.hu

Gyorsuló Ultrastar

A *Hitachi Global Storage Technologies* bemutatta *Ultrastar 15K147* sorozatú, nagyteljesítményű, vállalati alkalmazásokhoz szánt merevlemezeit. A 15000 1/perc fordulatszámmal üzemelő meghajtók a hasonló alkalmazásokban jelenleg leginkább elterjedt 10000 1/perc fordulatszámú egységekhez képest 33 százalékkal nagyobb teljesítményt nyújtanak, elérési idejük mindössze 2 ms, egyenletes teljesítőképességükhöz pedig a 16 MB méretű, ma még szokatlanul nagy gyorsítótár is hozzájárul. A legújabb *Ultrastarok* 36, 73 és 147 GB-os kapacitással kaphatók, csatolófelületük *Ultra320 SCSI* vagy 2 Gb/s sebességű *FC-AL* lehet.

➲ www.hgst.com

Hídfőállás Budapesten

Az indiai *Satyam Computer Services* új szolgáltató központot nyitott *Budapesten*. A cég hazánkból kedvező árfekvésű – magyarul olcsó – szolgáltatásokat kíván nyújtani nyugat-európai ügyfeleinek. A *Satyam Hungary Development Center* az *InfoParkban* bérel irodát, eleinte 60 főt foglalkoztat, ám a későbbiekben a létszámot további több százzal tervezik bővíteni. A budapesti letelepedés mellett két fontos érv szólt: a nyugat-európai ügyfelek közelsége és a jól képzett szakemberek viszonylagos olcsósága. Igaz, hogy egy indiai informatikus 7500 dolláros átlagos keresetéhez ké-

Satyam
What Business Demands.

pest egy magyar programozó 10500 dollárt keres, ám ugyanezért a munkáért Írországban 24500, az Amerikai Egyesült Államokban pedig 65000 dollárt is kifizetnek. A *Satyam* nem az első itthon megtelepedő tanácsadó-szolgáltató cég, hasonló területen mozog a *Tata Consultancy Services*, az *EDS* és az *IBM* is.

➲ www.satyam.com

Gazdaságos zenék

Az internetes zeneletöltések váratlan sikere egyre több szereplőt vonz erre a területre. A legújabb a hazánkban is

TESCO

jelen lévő *Tesco*, amely a 25 millió fontosra becsült angliai piacból szeretné kihasítani a maga szeletét. A *Windows Media Playerrel* együttműködő internetes áruházban több mint félmillió jó minőségű zeneszámot terveznek elérhetővé tenni, a zeneszámok darabjáért 79 pennyt, vagyis körülbelül 280 forintot kell majd fizetni. Ezzel az árral a *Tesco* viszonylag versenyképes lesz, a többi szereplő ugyanis – az *iTunes* kivételével – vagy magasabb árat kér, vagy kisebb választékkal szolgál.

➲ www.tesco.com

QNX: eladták, de marad

A *Harman International Industries* bejelentette, hogy november végi hatállyal felvásárolja a valós idejű operációs

QNX
QNX SOFTWARE SYSTEMS

rendszeréről ismert kanadai *QNX Software Systemst*. A *QNX* részéről érkezett nyilatkozatok szerint a két cég termékei kiválóan kiegészítik egymást, a *Harman* ugyanis hangrendszereket és telematikai elektronikus készülékeket gyárt az autóipar számára, míg a *QNX* beágyazott operációs rendszerével komoly beszállítói eredményeket ért el ezen a területen. A *Harman* állítólag meghagyja a *QNX* fejlesztői részlegének önállóságát, sőt, az operációs rendszer a *Harman* versenytársai – ilyen a *Delphi* és a *Visteon* – számára is megvásárolható marad. A *Harman* egyébként több mint egymilliárd dollár értékben szállít autós rendszereket többek közt a *Chryslernek* és a *Mercedes-Benznek*, mellette egészen eltörpül a mindössze 25 millió dolláros bevételt felmutató *QNX*. A felvásárlás ugyanakkor rámutat, hogy az autóiparban is egyre nagyobb szerephez jutnak a szoftverek, márpedig az autógyárak is átfogó megoldásokat várnak beszállítóiktól.

➲ www.harman.com
➲ www.qnx.com

Kiképzés

Márciusig a *Novell* több mint 500 platina szintű – illetve, ha ők nem töltik meg a felkínált helyeket, akkor arany szintű – értéknövelő viszonteladójának ingyenes *Novell Certified Linux Professional* képzést biztosít. Az új minősítő program a *Novell SuSE Linux Certified Professional* programjának megújított változata, amely a közeljövőben fokozatosan átveszi elődje helyét. A novemberben induló tanfolyamok öt naposak, díjuk normál esetben kereken hatezer dollár.

➲ www.novell.com/training/certinfo/clp

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

Mi újság a rendszermag fejlesztése körül?

A *Red Hat* képviselői úgy döntöttek, hogy az általuk nemrég megvásárolt *Global Filesystem (GFS)* fürtöző fájlrendszert *GPL* hatálya alatt teszik elérhetővé. A tervezet múltja meglehetősen kalandos. Először mint a *Sistina GPL*-es fejlesztése indult, ám a cég 2001-ben *Sistina Public License* hatálya alá helyezte át, ami a forráskód továbbadását díjfizetéshez kötötte. A döntést hatalmas felzúdulás követte, többek közt *Alan Cox* is kifogásolta, hogy a szerződésváltás saját szerzői jogait is sérti, hiszen maga is hozzájárult a *GFS* fejlesztéséhez. Elindult az *OpenGFS Project*, amely a *Sistina*-féle kód utolsó *GPL* hatálya alá tartozó változatára épült. A következő években a *Sistina* megpróbálta saját termékeként terjeszteni a *GFS*-t, 2003-ban még a *CommVaulttal* is szövetségre lépett. 2004-ben a *Red Hat* megvette a kódot a *Sistinától*, és most újra *GPL* hatálya alatt jelentette meg. Annak idején, 2000-ben a *GFS* jó eséllyel bekerülhetett volna a hivatalos *Linux* rendszermagba. Most, hogy újra elérhető, a *Red Hat* is támogatja a beépítését. Jelenleg úgy néz ki, hogy a kód szabad marad.

Linus Torvalds új folttulajdonlási rendszerre tett javaslatot, amely a jelek szerint egyelőre sikeres is. A cél, ahogy ő mondja az, hogy a rendszermag fejlesztői esetleges szerzői jogsértésre vonatkozó vád esetén követhessék a foltok sorsát. *Linus* indokai egyértelműek. Ő és segítői rengeteg időt fordítottak arra, hogy a *The SCO Group* szerzői jogsértésre vonatkozó vádjait elhárítsák. Ehhez – legalábbis eddig – ősrégi levelezési listák anyagán kellett átrágniuk magukat. A *Linus* által javasolt tulajdonlási rendszer, ami úgy tűnik, rövid egyeztetések után széles körű alkalmazásra kerül, mindössze annyit tesz szükségessé, hogy a fejlesztők feltüntessék nevüket a foltokban, és jelezzék egyetértésüket a rendszermagra vonatkozó szerződéssel. Minden foltban szerepelni fog minden olyan fejlesztő neve, aki a rendszermagba való bekerülése előtt módosította. Ezzel a megoldással a szerzői jog későbbi állítólagos megsértéseit csupán a foltok áttekintésével ki lehet vizsgálni, és nem lesz szükség arra, hogy a foltok történetét a levelezési listák bújásával próbálják visszakövetni. Írásom születésekor sok fejlesztő már használja az új rendszer, szerencsére eddig nem volt szükség az így összegyűjtött adatokra.

Randy Dunlap továbbfűzte az a meglehetősen régi ötletet, amely szerint a *.config* adatokat fordítás után magába a rendszermagba kellene elmenteni. Szerinte a rendszermag változatszámát, továbbá a fordítás dátumát is fel kellene jegyezni. A *.config* adatok rendszermagba való mentésének ötlete annak idején sok vitát váltott ki, hiszen ezeket a rendszermagon kívül is megfelelően tárolni lehet. Mivel e tekintetben sikerült egyezségre jutni, *Randy* újabb adatok beépítésére vonatkozó javaslata jóval kedvezőbb fogadtatásra talált. Sőt, több fejlesztő is megjegyezte, hogy az ötletet már jóval korábban meg kellett volna valósítani. Úgy látszik, az emberi természet alapvető jellegzetessége, hogy még a legnyilvánvalóbb dolgokat is fel kell fedeznie valakinek.

A *Jeff Dike*-féle *User-Mode Linux (UML)* műszaki nehézségek miatt nehézkesen halad a hivatalos 2.6-ös rendszermagfa részévé válás irányába. Látszólag *Andrew Morton* több mint boldogan fogadja *Jeff* foltjait, csakhogy *Jeffnek* gondjai vannak a foltok elfogadható darabokra osztásával. *Jeff* maga is bevallotta, hogy az *UML*-es munka kapcsán „kiszúrt magával", ugyanis nem tudja, milyen eszközökkel kezelhetné a folt felosztását. Az *UML* folt annyira nagyra nőtt, hogy feldarabolása komoly kihívássá vált. A nagyobb bővítésekkel általában ez történik. Sokszor szükségtelenül sok energiát fordítanak a folt beépítésének előkészítésére, végül amikor a fejlesztők elérkezettnek látják az időt, hirtelen rádöbbennek, hogy még rengeteg egyéb munka áll előttük, és a folttal még nem is foglalkoztak. A vége természetesen végeláthatatlan vitatkozás. *Jeff* számára nem újak a felmerülő kérdések, de hiába van tisztában a teendőkkel, ha a foltokat rendkívül nehéz egy-egy feladatot ellátó apró részekre osztani. Az biztos, hogy az *UML* be fog kerülni a 2.6-os rendszermagba, de a nehézségek miatt komoly csúszásra kell számítani.

A *CREDITS* fájl karbantartását sokáig *John A. Martin* végezte, és tevékenységéért a *MAINTAINERS* fájlban is elismerésben részesült, ám most úgy látszik, hogy a *CREDITS* fájl önkarbantartóvá vált, külön gondozására nincs többé szükség. *Linus Torvalds*, *Andrew Morton* és a rendszermag többi karbantartója gyakorlatilag átvették ezeket a feladatokat, a fejlesztői foltok sokszor saját frissítést tartalmaznak a *CREDITS* fájlhoz, ami annak külön bővítését szükségtelenné teszi. A *CREDITS* fájllal tehát egyszerűen az történt, hogy a rendszermag fejlesztésének teljes jogú részévé vált. Annak idején, első összeállításakor meglehetősen csüggesztő feladat volt a fájlt gondozni, ugyanis nagyon sok hozzájáruló neve kimaradt belőle. Most, hogy önfenntartóvá vált, tartalma is lényegesen pontosabb. *John* úri eleganciával lépett le, amikor *Adrian Bunk* rámutatott, hogy karbantartói munkája immár felesleges, ugyanakkor azt még megjegyezte, hogy ha a jövőben újra szükség volna a segítségére, akkor kész ismét munkába állni.

Zack Brown

Linux Journal 2004. november, 127. szám

Új termékek

Linux 101

A klasszikus IBM billentyűzetek felépítésén alapuló, testreszabott billentyűzeteket gyártó és javító

cég, az UniComp bejelentette a Linux 101 billentyűzet megjelenését. A Linux 101 egy programozott billentyűzet, ami a Linux felhasználók számára a kényelmesebb elérhetőség érdekében átrendezi a Ctrl, Caps-Lock és Esc billentyűk elhelyezkedését. A Linux 101 billentyűzet kapható az alapjaként szolgáló IBM M modellből származó „buckling spring" mechanikával, vagy a csendesebb gumimembrános felépítésben. Az UniComp honlapjáról letölthetőek az egyes elrendezéseket bemutató PDF-fájlok.
www.pckeyboard.com

NetDirector Linux Configuration Suite

Az Emu Software cég NetDirector Linux Configuration Suite programcsomagja egy méretezhető többkiszolgálós felhasználásra tervezett vezérlőeszköz, amely vegyes linuxos környezetben képes a különböző gyártóktól származó és eltérő tevékenységeket végző kiszolgálók hálózatának összefogására. A NetDirector csomag felhasználói felületén keresztül a rendszergazda központilag felügyelheti az Apache, Samba, DNS, DHCP, FTP vagy levelezés szolgáltatásokat nyújtó kiszolgálók működését. Lehetőség van a kiszolgálók szervezetek, földrajzi elhelyezkedés, vagy a kiszolgálón futó alkalmazás szerinti csoportosítására és kezelésére. Az intézkedések ezután alkalmazhatók a teljes csoportra, vagy az elrendezés meghatározott kiszolgálójára.
wwww.emusoftware.com

Enea Orchestra

Az Embedded Linux és az Enea cég szigorúan valósidejű (hard real-time) operációs rendszere képezi az alapját az Enea Orchestrának, ennek a magas rendelkezésre állással rendelkező távközlési és adatkommunikációs programok számára készült programfelületnek. Az Enea Orchestra lehetővé teszi a távközlési és adatkommunikációs készülékek gyártói számára, hogy magas rendelkezésre állással és nagy hibatűréssel rendelkező többprocesszoros programrendszereket helyezzenek üzembe. Az Orcestra tartalmazza a Metrowerks Corporationtől származó beágyazott fejlesztői technológiát, amely egyszerűsíti a rendszermag ellenőrzését, az áramköri kártyák hibáinak felmérését, az alkalmazások létrehozását és tesztelését. Az így létrejövő programok futtathatók bármilyen Linux-változaton, az OSE alatt, vagy a kettő valamilyen kombinációján. Az Enea Orchestra előfizetői rendszerben hozzáférhető és tartalmazza az alkalmazásfejlesztői és a felületfejlesztői csomagokat.
www.enea.com

Motorola A780

A Motorola A780 mobiltelefonjának alapját a Linux és Java programok képezik. A lenyitható for-

májú telefon szolgáltatásai közt egyebek mellett megtaláljuk a PDA-szerű, egynegyed VGA-kép megjelenítésére képes (240 x 320 képpontos) színes érintőkijelzőt, a Bluetooth hálózati és szinkronizációs szolgáltatást, egy 1,3 megapixeles digitális fényképezőgépet, MP3 lejátszót, 48 MB memóriát vagy eltávolítható TransFlash tárolót. Az A780 négysávos GSM telefon, így támogatja az elterjedt amerikai és nemzetközi frekvenciákat. A telefon tartalmazza az EDGE technológiát is, amely akár 240 Kbps sebességű Internetes elérést biztosító adatátvitelt is lehetővé tesz.
www.motorola.com

EtherDrive tárolókártyák

A Coraid cég piacra dobta az EtherDrive Storage Blades nevű tárolókártyáit, amelyek méretez-

hető hálózatos blokktároló létrehozását teszik lehetővé az AoE (ATA-over-Ethernet) protokollt használó kiszolgálók számára. Az EtherDrive tárolókártya a szabványos ATA meghajtókat fogja össze egységes, rugalmasan alkalmazható, keretekbe szerelhető tárolóegységgé. Minden EtherDrive tartalmaz saját Ethernet csatolót és nanokiszolgálót, amelyek az Ethernet-ATA protokoll-átalakítást végzik. Az EtherDrive meghajtók 250 GB-tól 16 petabájt méretig terjedő megosztott tárolási kapacitású egységek létrehozását biztosítják. Egyaránt hozzáférhető a szabványos 2,5 vagy 3,5 colos ATA meghajtókkal. A Linux 2.4 és 2.6 rendszermagokhoz már forráskódban hozzáférhetőek GPL licencű meghajtók, egyéb operációs rendszerekhez pedig hamarosan elkészülnek.
www.coraid.com

Quadputer-Navion

A Microway cég Quadputer-Navion nevű gépe egy SMP (Symmetrical Multi-Processing, szimmetrikus többprocesszoros) kiszolgáló négy AMD Opteron 850 processzorral, amely SuSE Linuxot futtat. A 4U készülékházban elhelyezkedő Quadputer-

Új termékek

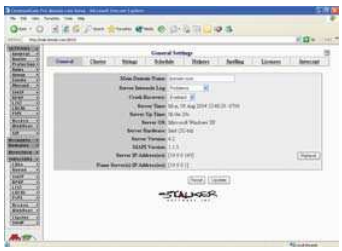
Navion 810 wattos redundáns, hot-swap (menet közben cserélhető) tápegységgel rendelkezik, és lehetővé teszi 18 darab SATA/IDE vagy 5 darab SCSI meghajtó beépítését. A processzorok működési frekvenciája 2,2 GHz és támogatják a HyperTransport technológiát.
A 2,5" méretű eszközökkel akár 1,36 TB tárolókapacitású beépített RAID-rendszer hozható létre. A konfiguráció tartalmazza a Microway Nodewatch és MCMS hardver- és szoftver-kezelő eszközeit amelyek lehetővé teszik a fürtözött gépek távoli ellenőrzését és vezérlését.
www.microway.com

SC-4400 FlashDisk OpenRAID

A Winchester Systems bejelentette a FlashDisk OpenRAID-et, amely egy akár öt különböző típusú kiszolgáló támogatására is képes adattároló tömb. A kiszolgálók a tárolóeszközökön U160 SCSI protokoll segítségével osztoznak, a rendszer legfeljebb 2,3 TB tárhely kezelésére képes. További bővítő készülékház hozzáadásával a FlashDisk OpenRAID 4,6 TB kapacitásig méretezhető, legfeljebb három állománnyal. Támogatja a kettős, redundáns, 64 bites RAID-vezérlőket, és különféle a lemezeket erősen terhelő feladatokhoz használható, például adatbázisok kezelésére, multimédiás alkalmazásokhoz, elektronikus levelezési vagy webkiszolgáló rendszerekhez. Az SC-4400 minden kiszolgáló operációs rendszerrel képes együttműködni, a gazdaoldalon illesztőprogramot nem igényel.
www.winsys.com

CommuniGate Pro 4.2

A 4.2-es változat megjelenésével a CommuniGate Pro Real-Time Communications immár biztonságos azonnali üzenetküldést, VoIP-támogatást, videokonferencia lehetőséget, rajztábla-megosztást, valamint asztal- és alkalmazásmegosztási lehetőséget egyaránt biztosít. A szabványokra alapuló, többféle géptípuson való használatot célzó felépítésnek és a SIP protokollnak köszönhetően a CommuniGate Pro 4.2 tetszőleges ügyfélen futtatható, akár az irodában, akár tá-



volról, UNIX, Linux, Mac OS X és Windows operációs rendszeren. Ugyancsak a 4.2-es változat újdonsága a teljes mértékben testreszabható webes felület és a webes levelezési és naptárkezelési lehetőség. Mindemellett a CommuniGate Pro 4.2 hibaelhárítási, karbantartási célokra használható távoli felügyeleti szolgáltatásokat is biztosít.
www.stalker.com

Wyse Winterm 5150SE

A Wyse Winterm 5150SE egy Wyse Linux V6 operációs rendszerre és AMD Geode GX 533 processzorra épülő vékony ügyfél. A Wyse Linux V6 a 2.6-os Linux rendszermagra alapul, testreszabási lehetőségei révén számos különböző környezetben képes megfelelni az igényeknek, ide értve a Windows, a UNIX, a Linux, az IBM, az X-Window és a Java alapú rendszereket. A Winterm 5150SE szabad munkaállomás-váltási lehetőséget kínál, vagyis a felhasználók bármelyik ügyfélre bejelentkezhetnek, beállításaik bárhol elérhetők maradnak. Az 5150SE moduláris felépítésének köszönhetően a szükséges bővítések könnyedén elvégezhetők. A vékony ügyfél csak olvasható fájlrendszerrel rendelkezik, mozgó alkatrészeket nem tartalmaz, kisméretű készülékháza pedig USB és hagyományos be/kiviteli kapukkal egyaránt rendelkezik.
www.wyse.com

Atigo felölthető számítógépek

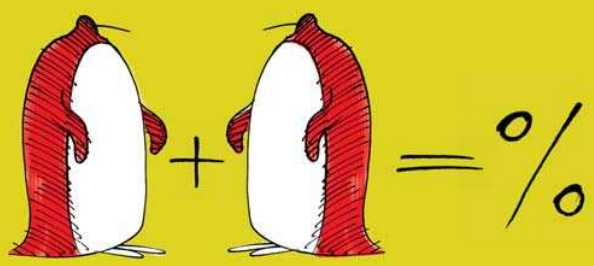
A Xybernaut bejelentette, hogy Atigo vezeték nélküli panelgépei már Linuxszal is elérhetők. Az Atigo gépek vezeték nélküli, lapos képernyős számítógépekként vagy önálló, vezeték nélküli kapcsolatok kezelésére képes mobil/felölthető számítógépeként használhatók. Az Atigok támogatják a kettős használatot, az IEEE 802.11b szabvány szerinti vezeték nélküli hálózatokat normál PC-kártya és/vagy CompactFlash kártya segítségével érik el. A linuxos Atigok nyílt forrású támogatási, adatátviteli, adatkezelő és rendszerbeállító segédeszközöket is tartalmaznak. Az Atigo T modell 1 GHz órajelű Crusoe TM5800 processzort és 256 MB SDRAM memóriát tartalmaz, Flash memó-

riája 128, 256 vagy 512 MB vagy 1 GB méretű lehet. Minden Atigo belső, újratölthető lítium-ion akkumulátorral rendelkezik, de üzem közben cserélhető külső akkumulátorokkal is használható. A gépek mindegyike 8,4"-es, 800 × 600 képpont felbontású SVGA kijelzővel rendelkezik.
www.xybernaut.com

Linux Journal 2004. 126. és 127. szám

Linux-index

1. A linuxos kiszolgálók kínai piacának a következő öt évre vetített százalékos növekedése: 49,3
2. Azoknak a fejlesztőknek a száma, akik rendszeresen nyújtanak be módosításokat a *Linuxhoz*: 1000
3. A fentiek ekkora százaléka kap fizetséget munkaadójától linuxos hozzájárulásáért: 10



4. Az utolsó 38000 *Linux*-módosítás ekkora százalékát adják a fizetett fejlesztők: 97,4
5. A *Linuxhoz* kapcsolódó termékek és szolgáltatások piaca 2003-ban, milliárd dollárban: 2,5
6. A *Dice.com* webhelyen szereplő számítástechnikusi állások becsült száma: 49
7. A *Dice.com* ajánlatai közül körülbelül ennyiszer száz számítástechnikusi álláshoz kellenek linuxos ismeretek: 22
8. Az elmúlt évhez képest ez az érték ennyi százalékkal nőtt: 190
9. A *Dice.com* ajánlatai közül ennyi számítástechnikusi álláshoz kellenek linuxos vizsgák: 10
10. Az *Apache* helyezése a *Netcraft* webkiszolgálókra vonatkozó felmérésében: 1
11. Az *Apache* százalékos részesedése az utolsó felmérés szerint (2004 augusztusa): 67,37

Források

1: *CCID Consulting*, az *Oracle* közreműködésével
2–4: *Andrew Morton*
5: *Morgan Reed, Association for Competitive Technology*
6–9: *Computerworld*
10, 11: *Netcraft, Ltd.*, ➲ *netcraft.com*

Linux Journal 2004. november, 127. szám

Ők mondták

Általános tévhit a *Linux* fejlesztőkkel kapcsolatban, hogy ezek olyan férfiak, akik szabadidejükben egy pincében ülve kódolnak, s mindezt tisztán a szakma szeretete okán. Az igazság az, hogy ezek az emberek valóban komoly hátteret jelentettek a *Linux* számára – nagyjából öt évvel ezelőttig.

Andrew Morton

➲ *gcn.com/vol1_no1/daily-updates/26641-1.html*

A nyíltságra két helyen van szükség. Az egyik a forráskód. A másik az adatkezelés.

Roml Lefkowitz

Igazgató, nyílt forrás, *AT&T Wireless* (beszéd az *O'Reilly Open Source* konferencián)

A történelem során még soha nem láthattunk olyat, hogy az aktuális helyzet élvezői régebbi megoldásokra épülő üzleti modelleket védtek volna az újabb megoldások alkotó jellegű rombolásától. Most a politika befolyásolásával mégis megteszik ezt. A telegráf nem akadályozhatta a telefon, a vasút nem gátolhatta az autó elterjedését. Most azonban, a lobbizásra költött hatalmas összegeknek köszönhetően és a média által igényelt szinte végtelen nagyságú pénzek miatt a politikai folyamatokat ezek a szereplők befolyásolják.

Howard Rheingold

➲ *www.businessweek.com/bwdaily/dnflash/aug2004/nf20040811_1095_db_81.htm*

A szabad szoftverek alkalmazása annak a kérdésnek a megoldására, hogy a beszállítók, szolgáltatók sokszínűsége közgazdaságilag, míg a termékek minél kisebb változatossága mérnökileg előnyös, ám a kettő összeegyeztetése nehezen megoldható, az egész iparban egyedülállónak mondható. Nem tudok hasonló példákat felhozni.

Andrew Morton

➲ *www.groklaw.net/article.php?story=20040802115731932*

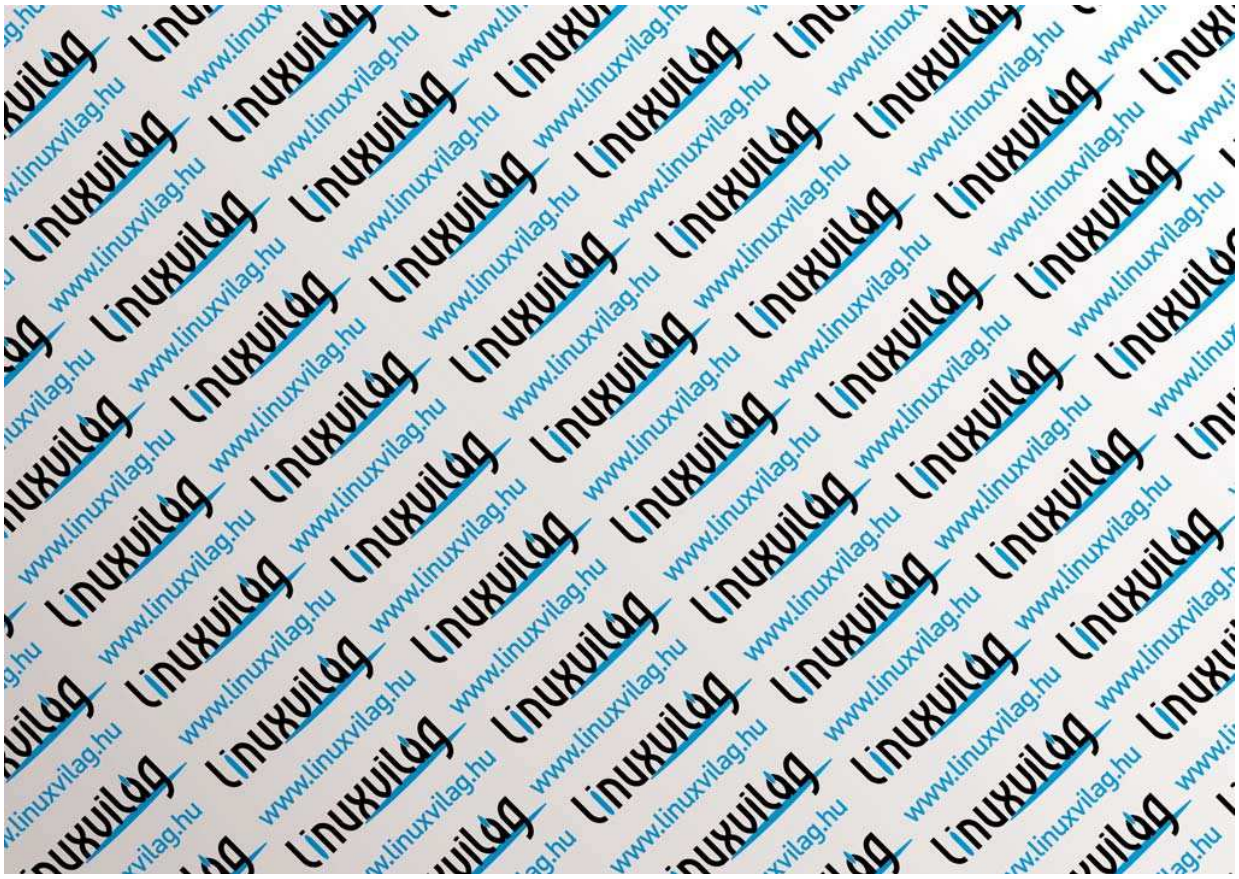
Linux Journal 2004. 127. szám

Nem lesz 2.7-es rendszermag?

Felejtsünk el mindent a rendszermagok páros-páratlan számozásáról.

A 2004-es ***Linux Kernel Summit*** rendezvényen a rendszermag vezető fejlesztői bejelentették, hogy a 2.7-es fejlesztői rendszermag a közeljövőben nem jelenik meg. Úgy nyilatkoztak, hogy megfelelt nekik a dolgok eddigi menete, és nem akartak változtatni rajta. Mivel a bejelentés rengeteg félreértést okozott, megpróbálom elmagyarázni, mi is történt.

A 2.5-ös rendszermag fejlesztése közben a felső szintű karbantartási folyamatok megváltoztak. Mielőtt ***Linus*** elkezdte volna használni a ***BitKeeper*** változatkövető rendszert, a karbantartók egyszerre 10–20 foltot is küldtek neki, majd meg kellett várniuk a következő kiadást, csak akkor derült ki, hogy kódjuk bekerült-e a rendszermagba. Ha nem, újra kellett próbálkozniuk. Ez a rendszer több mint tíz éven keresztül remekül megfelelt. A 2.5-ös sorozat fejlesztésének megkezdése után hatalmas perpatvar kerekedett a kihagyott foltok miatt, ami azzal végződött, hogy ***Linus*** a ***BitKeeper*** használata mellett döntött. Miután a ***BitKeeper*** fejlesztőivel nem kis munkát fektettek a rendszermag fejlesztői által igényelt szolgáltatások megvalósításába, ***Linus*** a 2.5.3-as rendszermag 2002. február 2-i kiadásakor bejelentette a ***BitKeeperre*** való áttérést. A fejlesztők túlnyomó részének munkája semmit sem változott. Továbbra is elküldték kódjaikat a felsőbb szintű karbantartóknak, majd vártak. A karbantartók egy kis részének, akik a ***BitKeeper*** használata mellett döntöttek, azonban gyökeresen átalakult az élete.



Nekik létre kellett hozniuk egy ***BitKeeper*** fát, fel kellett tölteniük a ***Linusnak*** szánt módosításokkal, majd jelezniük kellett neki. ő átemelte a foltokat saját fájába, és elrendezte a mások munkáival kialakuló kisebb ütközéseket.
A ***BitKeeper*** használatának nem várt következményei is voltak. Először is, bármely pillanatban bárki figyelemmel követhette ***Linus*** fájának állapotát. Néhány fejlesztő, köztük ***Peter Anvin*** és ***Jeff Garzik*** megoldották, hogy az éjjelenkénti pillanatfelvételek a ***kernel.org***-on is megjelenjenek.
A ***BitKeeper*** fejlesztőivel együttműködve ***CVS*** és ***Subversion*** tárolókat is készítettek a változatkövető rendszert használóknak.
A fa aktuális állapotának ismerete lehetővé tette, hogy a karbantartók gyorsabban eljuttassák foltjaikat ***Linusnak***, továbbá azt is láthatták, hogy mikor fogadta el őket. Nem kellett két hetet várni egy-egy pillanatfelvételre, azonnal el lehetett küldeni a módosításokat. A rendszermag fejlesztése azonnal felgyorsult.
Másodszor, minden ***Linus*** fájába bekerült foltról értesítés ment egy levelezési listára, ennek segítségével bárki követhette a változásokat. A fejlesztők figyelemmel kísérhették a rendszermag tetszőleges részét érintő módosításokat, megtudhatták azok okát, és rámutathattak az esetleges problémákra. A lista révén javult a kölcsönös ellenőrzés, a hibákat hamarabb jelezni lehetett, az egyes fejlesztési területekkel kapcsolatosan pedig mindenki naprakész információkkal rendelkezhetett.

Végül elkészült a 2.6

A rendszermag fejlesztői 2002. december 31-én jelentették be a 2.6-os sorozat szolgáltatáskészletének befagyasztását. 2003. július 7-én napvilágot látott az első ***2.6.0-test1*** rendszermag, a karbantartási folyamat pedig újfent átalakult. ***Linus*** helyett ***Andrew Morton*** fogadta szinte az összes foltot. A ***BitKeepert*** használók fáit viszont továbbra is közvetlenül Linus vette át. Végül 2003. december 17-én megjelent a 2.6.0-s rendszermag – a 2.5-ös és a 2.6-os változat fejlesztésének 680 napja alatt óránként átlagosan 1,66 módosítást végeztek el.
A 2.6.x sorozat következő öt tagját nagyjából havonta adták ki, mindegyik kiadás 538-1472 módosítás eredményeként született meg. Ezután a 2.6.5-ös változattal felgyorsultak a dolgok, a 2.6.6-os változat 1757, míg a 2.6.7-es 2306 módosítás nyomát viselte magán. A 2.6.0 - 2.6.7 időszakban az üzembiztos rendszermag óránként átlagosan 2,2 foltot olvasztott magába, vagyis gyorsabban változott, mint a fejlesztői változat. A 2.6.7-es ugyanakkor számos teljesítménymérés és teszt szerint minden idők legstabilabb ***Linux*** rendszermagja volt.
Talán a fejlesztők megbolondultak, és csak úgy, kényük-kedvük szerint kezdtek el ellenőrizetlen kódrészleteket kiadni? Nem. A 2.6-os sorozatban ***Andrew Morton*** továbbra is ellenőrzött minden benyújtott foltot, mielőtt továbbadta volna őket ***Linusnak***. A ***BitKeepert*** használó fejlesztők továbbra is ***Andrew*** fájában ellenőrizhették foltjaik állapotát, és ha nem merült fel semmi gond, ***Linustól*** kérhették elfogadásukat.
A módosításokat tehát jelenleg a ***-mm*** fa felhasználói ellenőrzik. Ez teljesen eltér a dolgok korábbi menetétől. Jelenleg a foltok ellenőrzését, fordítását, jó és rossz célú felhasználását a földkerekség felhasználói végzik, mielőtt véglegesen elfogadnák őket. Ha egy folttal vagy foltok egy csoportjával valamilyen gond van, ***Andrew*** egyszerűen kidobja őket saját fájából, és felkéri az eredeti fejlesztőt a javításra.
Mivel lehetőség nyílt arra, hogy a foltok szélesebb körű tesztelésen essenek át, mielőtt a fába bekerülnének, a 2.6-os sorozat fejlesztésének folyamata a következőkből fog állni:
Linus kiadja a következő 2.6-os rendszermagot.
A karbantartók elárasztják ***Linust*** a ***-mm*** fában már kipróbált foltokkal. Néhány hét elteltével ***Linus*** kiad egy ***-rc*** rendszermag-pillanatfelvételt.
Mindenki megpróbál magához térni a változtatások özönétől, majd kijavítani a ***-rc*** rendszermagban talált hibákat.
Néhány héttel később megjelenik a következő 2.6-os rendszermag, és a folyamat újrakezdődik.
Előfordulhat azonban, hogy túl sok változás áll be ahhoz, hogy kezelni lehessen őket, ekkor akár a 2.7-es rendszermag ág létrehozására is sor kerülhet. Ezt ***Linus*** feladata megtenni, az új, még kipróbálatlan foltok ebbe a fába kerülnének. Ezután, amint a 2.7-es rendszermag üzembiztossá válik, sort kerít a 2.6-ost érintő, folyamatban lévő módosítások átvételére a 2.7-esbe. Ha úgy ítéli meg, hogy a 2.7-es rendszermag fejlesztése rossz irányt vett, akkor a 2.7-est törölni fogják, és mindenki a 2.6-os fán fog tovább dolgozni. Ha a 2.7-es fa stabillá válik, akkor vagy visszakerül a 2.6-osba, vagy megkapja a 2.8-as számot.
A sok bonyodalom oka az, hogy a rendszermag fejlesztői a jelenlegi helyzetben kiválóan együtt tudnak működni.
A komolyabb változások, mint például a 8 kB-os rendszermagvermek bevezetése a 4 kB-os helyett, az üzembiztos sorozatot érintik. A felhasználók így viszonylag hamar igénybe vehetik a legújabb szolgáltatásokat. A terjesztések készítői üzembiztosabb rendszermagokat biztosíthatnak ügyfeleiknek, hiszen nincs szükség arra, hogy a fejlesztői ágból külön munka árán átemeljék a foltokat az üzembiztos ágba, ahogy azt a 2.5-ös sorozatnál is láttuk.
A gyorsabb fejlesztés miatt a rendszermagon belüli ***API*** gyakorlatilag folyamatosan változik. A ***Linux*** esetében ez nem újdonság, ám most még nagyobb hangsúlyt kap.
A fő ***kernel.org*** fában nem szereplő rendszermagmodulok tehát rendkívül gyorsan működésképtelenekké válhatnak. Fontos tehát, hogy ezek a modulok megtalálhatók legyenek a fő rendszermag fában. Így az ***API***-ra vonatkozó módosításokat az érintett modulokban is végrehajtják, és minden felhasználó élvezheti a modulok által biztosított szolgáltatásokat.
A folyamat valójában nem hirtelen változott meg, inkább azt mondhatjuk, hogy lassacskán kifejlődött valami kiválóan működő dolog – sőt, a rendszermag fejlesztőinek közösségén kívüli sokan talán észre sem vették, hogy bármi is változott, egyszerűen csak rádöbbentek egy nap, hogy minden korábbinál jobb ***Linux*** rendszermag fut a gépükön.

Linux Journal 2004. november, 127. szám

Greg Kroah-Hartman (greg@kroah.com)
Jelenleg a Linux-rendszermag különféle illesztőprogram-alrendszereiért felelős. Az IBM-nél dolgozik és a Linux-rendszermaggal kapcsolatos kérdésekkel foglalkozik.

Gyors alkalmazásfejlesztés Python és Glade használatával

A Glade segítségével könnyedén elkészíthetjük és módosíthatjuk Python alkalmazásaink grafikus felületét, sőt, még az eseménykezelők létrehozását is önműködővé tehetjük vele.

A grafikus programok készítése két lépésből áll. Először meg kell írni a felületet létrehozó kódot, amely megjeleníti a különféle kezelőelemeket, menüket, gombokat, feliratokat. Ezután el kell készíteni az események bekövetkezésekor – például gomb lenyomásakor vagy menüelem kiválasztásakor – lefutó kódrészleteket. A program elinduláskor egy eseményhurokba lép be, várja az eseményeket, majd gondoskodik a megfelelő eseménykezelő meghívásáról. Ezt az adott eseményhez tartozó visszahívó függvénynek, röviden visszahívónak nevezünk. Kell például írnunk egy olyan függvényt, amely a gombok megnyomásakor fut le. A kezelőelemek megjelenítését végző kód megírása, az eseménykezelő függvények megadása és az eseményeknek a függvényekhez való csatolása túlságosan hosszadalmas és unalmas munka ahhoz, hogy kézzel végezzük el.

Sok kezelőelem-készlethez létezik olyan eszköz, amellyel grafikus felületen állíthatjuk össze programunk felületét. Damon Chaplin ***Glade*** néven készített egy ilyent a GTK/GNOME készlethez. Ezzel nemcsak a kezelőfelület képét alkothatjuk meg, hanem egyszerűen adhatjuk meg vele az egyes eseménykezelő függvényeket is. A ***Glade*** a kezelőelemeket és a visszahívókat egy XML fájlban tárolja, végeredményként pedig olyan C vagy C++ programot állít elő, amely a kezelőelemek megadott elrendezésének megfelelő hívások mindegyikét tartalmazza, továbbá létrehozza a visszahívók kapcsolatait és magukat az üres visszahívókat is.

Maga a ***Glade*** sajnos nem használható Python kód létrehozására, az általam írt ***GladeGen*** azonban a ***Glade*** XML fájl alapján Python kódot készít.

Ha megváltoztatjuk a grafikus felületet a ***Glade***-del, akkor a felület létrehozásához új C/C++ kódot kell készíttetni vele. Ezt ismételgetni elég unalmas lehet, különösen akkor, ha a grafikus felületet előállító kódot módosítottuk. Éppen azért készítette el James Henstridge a ***libglade***-et, amely lehetővé teszi, hogy a grafikus felületet előállító kódot ne kelljen bedrótozni az alkalmazásokba. James hozta létre a GTK/GNOME Python kötéseket, azt a Python modult, amely a GTK/GNOME C függvények elérhetőségét biztosítja.

A ***libglade*** használatakor programunk nem tartalmazza a grafikus felületet létrehozó kódrészleteket. A ***libglade*** ehelyett a program futásakor dolgozza fel az XML fájlt, és futási időben hozza létre a megadott felületet. Amikor tehát a ***Glade*** segítségével változtatunk valamit a grafikus felületen, nem kell módosítanunk az azt létrehozó kódot is.

Jómagam szeretek Pythont használni, hacsak nem olyan kódot kell írni, amiben sok a számítási művelet, vagy fontos a gyors futás. A Python magas szintű adatszerkezetei és értelmezőkörnyezete rendkívül gyors kódfejlesztést, karbantartást és módosítást tesznek lehetővé. A Linuxvilág 2003. októberi számában már megjelent egy bevezető jellegű cikk a ***PyGTK***-ról és a ***Glade***-ről, amely a ***Gtk*** és a ***Glade*** használatában járatlanok számára kiváló segítség az első lépésekhez.

Feleségem egy szemvizsgálattal és optikával foglalkozó vállalkozásnál dolgozik. Éppen nekik készítettem egy adatbázis-kezelő és számlázó rendszert, amikor elhatároztam a ***GladeGen*** megírását. Mielőtt a feleségem odakerül volna, egy Microsoft Access alapú rendszert használtak, amit ki tudja, ki fejlesztett ki. Mivel az illető már kilépett a cégtől, új rendszert akartak bevezetni. A város más hasonló vállalkozásaival is felvették a kapcsolatot, és érdeklődtek, hogy milyen programot érdemes használni. Kiderült, hogy az összes többi helyen drága, hibáktól hemzsegő programokkal küszködnek. Végül sikerült meggyőznöm a tulajdonost, hogy a nyár folyamán, nagyjából egy új program áráért el tudnék készíteni egy egyedi rendszert, ami pontosan azt nyújtja, amire szükségük van. Az egyetlen kérésem az volt, hogy Linuxot használhassak.

Így készült el egy ***Python/GTK*** alapú, az adatok tárolására ***PostgreSQL***-t használó program, amely minden keresési és táblázatkezelő műveletet SQL-parancsok segítségével végez el. A ***Python*** kód a ***PostgreSQL*** adatbázis és a ***GTK*** felület közötti kapcsolatot adja. A ***GTK*** és a ***PostgreSQL*** egyaránt C-ben készült, vagyis mindkettő gyors. A ***Python*** kód a korszerű processzorokon szerencsére szintén elég gyorsan fut ahhoz, hogy a kettő közötti kapcsolatokat képes legyen kezelni. A ***PyGTK*** felület ***PostgreSQL*** adatbázissal kombinálva tetszőleges adatbázis-kiszolgáló elérését lehetővé teszi. A grafikus kezelőfelület több számítógépen is futhat, míg az adatbázis-kiszolgáló közös. Az ügyfél/kiszolgáló felépítésnek köszönhetően a grafikus felület bárhol futtatható, a ***PostgreSQL*** kiszol-

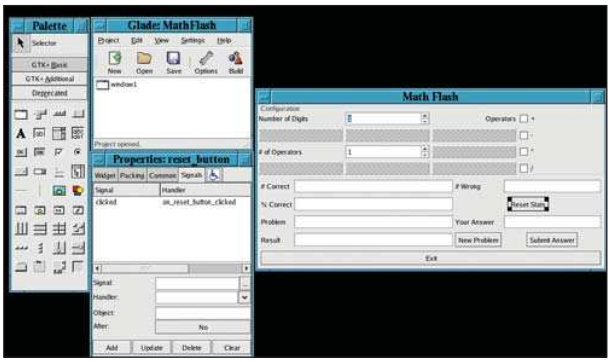
gáló pedig interneten keresztül, SSH-val titkosított kapcsolaton keresztül is elérhető. Az az előnye is megvan tehát ennek a megoldásnak, hogy távolról is bele tudok nyúlni a rendszerbe, ha egy felhasználónak valamilyen kérése van.
A program több mint negyvenféle ablakkal rendelkezik. Ezek többek közt a páciensek adatainak, a keretek és lencsék eladásainak, a kapcsolattartási adatoknak és a biztosítóktól érkező kifizetéseknek a bevitelét, illetve a keretkészlet követését teszik lehetővé. Úgy döntöttem, az ablakok létrehozására *Glade*-et használok, és mivel Pythonban akartam programozni, kellett írnom egy olyan programot, ami önműködővé teszi az ablakokhoz tartozó kód elkészítését. Az eredmény a *GladeGen*.

A GladeGen használata

Az általam írt kód önműködővé teszi a felületet összeállító Python kód elkészítését, megadja a visszahívó függvények kapcsolatait, biztosítja a kezelőelemek elérését és üres visszahívó függvényeket hoz létre a *Glade* XML fájl alapján. A program használatával egy grafikus felületű program létrehozása a következő lépésekből áll:

1) a felület grafikus összeállítása a Glade segítségével és az XML fájl elmentése
2) a *GladeGen* futtatása a kód előállítása végett
3) a visszahívók kódjának megírása.

A *GladeGen* által létrehozott kód minden az alkalmazás futtatásához és a felület megjelenítéséhez szükséges kódrészletet tartalmaz, illetve az üres visszahívó függvények is megtalálhatók benne. Lehetővé teszi, hogy a felületet a *Glade* használatával módosítsuk. Ha újra lefuttatjuk a *GladeGent*, újra létrehozza az alkalmazás kódját, amelybe az összes új visszahívó és kezelőelem bekerül, ám a meglévő kódot nem módosítja.
A továbbiakban egy matematikai kvízprogram elkészítésével fogom szemléltetni a *GladeGen* használatát. A teljes program a 65. CD Magazin/Glade könyvtárában található.
A *GladeGen* a *GTK 2.x* kezelőelem-készlettel és a megfelelő *Glade*-változattal használható, Red Hat rendszereken ennek *glade-2* a neve. Aki a *GTK* kezelőelemeket már ismeri, az a *Glade*-ben is otthonosan fog mozogni. Ellenkező esetben előbb ismerkedjünk meg a kezelőelemek tárolóosztályaival, de legalább a táblázatokkal és a vízszintes/függőleges dobozokkal.
A *glade-2* segítségével először elkészítettem a felület első változatát. Az első elem egy `GtkWindow` volt, amihez egy `GtkVBoxot` adtam hozzá. A függőleges dobozba két `GtkFrame` kezelőelem kerül, s mindkét keretbe egy-egy `GtkTable`-t illesztettem. A további kezelőelemeket a két táblázatba tettem. A feladványban szereplő műveleti jelek és a számjegyek számának kiválasztását `GtkSpinButton` kezelőelemekkel tettem lehetővé.
`GtkCheckButton` kezelőelemeket használtam annak jelzésére, hogy milyen műveleti jeleket kell használni a feladványban. A feladványokhoz, a válaszhoz és helyesen/hibásan megválaszolt feladványok megadásához `GtkEntry` kezelőelemeket alkalmaztam. Használtam még `GtkLabel` és `GtkButton` elemeket is.
A *Glade* fájlt `mathflash.glade` néven mentettem. A *Glade* nem kérdez rá, hogy akarjuk-e menteni a munkánkat, tehát



1. ábra Gomb visszahívójának megadása Glade alatt

ne feledkezzünk meg erről, ha valamilyen módosítást végeztünk a felületen. A felületet és magát a *Gladet* mutatja az 1. ábra. Látható, hogy a `reset_button gomb`, az `on_reset_button_clicked` függvényt hívja meg ha rákattintunk.
A *Glade* lehetővé teszi, hogy az egyes kezelőelemeknek nevet adjunk, illetve alapértelmezett neveket is előállít. A kezelőelemeknek, például a gomboknak és az adatbeviteli mezőknek igyekeztem olyan nevet adni, amely utal használatuk módjára. A *Glade*-ben azt is megadhatjuk, hogy a visszahívó függvényekhez milyen jelzéseket szeretnénk kapcsolni. Így határoztam meg azt is, hogy az `on_reset_button`-t kell meghívni, ha a felhasználó rákattint a `reset_button`-ra.
A következő lépésben a *GladeGen* segítségével, a `GladeGen.py mathflash.glade MathFlash MathFlash` paranccsal létrehoztam az alkalmazás sablonkódját. A parancssorban átadott értékek a *Glade* XML fájl neve, a létrehozandó *Python* fájl/modul neve, valamint az ebben a fájlban/modulban létrehozandó osztály neve. Az így kapott *MathFlash.py* fájl tartalmát mutatja az 1. kódrészlet.
A `GladeWindow` osztály `MathFlash` osztály alosztálya, amely a *libglade* segítségével gondoskodik a `handlers` (kezelők) változóban felsorolt visszahívók kapcsolatairól. Egyúttal egy szótárat is létrehoz *self.widgets* névvel, amely a `widget_list` változóban szereplő kezelőelem-neveket a megfelelő `GtkWidget` példányokhoz rendeli.
A `GladeWindow` alosztály alapértelmezett `show` (megjelenítő) és `hide` (elrejtő) eljárásokat is biztosít, így a sablonkód azonnal futtatható, és a felület a visszahívók megírásának megkezdése előtt ellenőrizhető.
A *GladeGenConfig.py* fájl lehetővé tesz bizonyos testreszabásokat, például a program készítőjének megadását, a `self.widgets` szótárba illesztendő elemtípusok kiválasztását és a létrehozandó *Python* kód kinézetének meghatározását. A megadott *GladeGenConfig.py* fájlban a `GtkWindow`, `GtkButton`, `GtkSpinButton`, `GtkCheckButton`, `GtkEntry`, `GtkCombo` és a `GtkTextView` elemtípusok szerepelnek.
A `GtkLabel` elem és a befoglaló elemek elérésére ritkán van szükség, ezért nem szerepelnek a `GladeGenConfig include_widget_types` listájában.
A létrejött *MathFlash.py* fájl az összes visszahívó függvényt tartalmazza, ezek törzse azonban egyelőre a Pythonban az üres műveletet jelző `pass` utasításból áll. Az eljárások megadása a `self` átadott értékkel, illetve a tetszőleges hosszúságú átadott értéklista kezelésére alkalmas `*args` mutatóval

1. kódrészlet A GladeGen ezt a Python kódot állította elő a Glade által készített XML fájl alapján

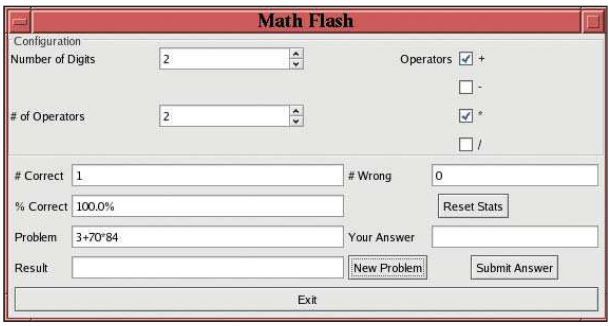
```
#!/usr/bin/env python
#---------------------
# MathFlash.py
# Dave Reed
# 02/28/2004
#---------------------
import sys
from GladeWindow import *
#---------------------
class MathFlash(GladeWindow):
    #--------------------
    def __init__(self):

        ''' '''


        self.init()
    #--------------------
    def init(self):
        filename = 'mathflash.glade'
        widget_list = [
            'window1',
            'plus_check',
            'minus_check',
            'multiply_check',
            'divide_check',
            'digits_spin',
            'operators_spin',
            'correct_entry',
            'wrong_entry',
            'pct_entry',
            'problem_entry',
            'answer_entry',
            'submit_button',
            'reset_button',
            'exit_button',
            'new_button',
            'result_entry',
            ]
        handlers = [
            'on_operator_check_toggled',
            'on_submit_button_clicked',
            'on_reset_button_clicked',
            'on_exit_button_clicked',
            'on_new_button_clicked',
            ]
        top_window = 'window1'
        GladeWindow.__init__(self, filename,
        top_window, widget_list, handlers)
    #--------------------
    def on_operator_check_toggled(self, *args):
        pass
    def on_submit_button_clicked(self, *args):
        pass
    def on_reset_button_clicked(self, *args):
        pass
    def on_exit_button_clicked(self, *args):
        pass
    def on_new_button_clicked(self, *args):
        pass
#--------------------
def main(argv):
    w = MathFlash()
    w.show()
    gtk.main()
#--------------------
if __name__ == '__main__':
    main(sys.argv)
```

történik. ***Python*** alatt a `*args` segítségével egy függvénynek/eljárásnak tetszőleges számú értéket lehet átadni, a hívó függvény ezeket tuple formájában kapja meg. A visszahívók jelentős hányada egyrészt az esemény által érintett kezelőelem azonosítóját, másrészt az esemény egyéb jellemzőit kapja meg, ha vannak ilyenek. A GTK weboldalon elérhető API leírásban minden visszahívó átadott értékei megtalálhatók.

Következő teendőnk a visszahívók kódjának megírása és a programhoz szükséges egyéb kódok elkészítése. Ennél a programnál nagyjából még 60 sornyi Python kóddal teljessé tehető a ***MathFlash.py*** fájl. Egy tapasztalt ***Glade***-használó és ***Python***-programozó nulláról indulva nagyjából fél óra alatt össze tudja állítani a teljes kódot. Az összetettebb programok írásakor a modell/nézet/vezérlés tervezési sorrendet érdemes követni. A ***GladeGen*** által előállított kód maga a vezérlés. Az `on_submit_button_clicked` visszahívó példáján vizsgáljuk meg, hogyan kell használni a ***GladeGen*** által előállított ***PyGTK*** kódot. Az általam írt ***Python*** kód a következő:

```
def on_submit_button_clicked(self, *args):
    prob =
        ➥self.widgets['problem_entry'].get_text()
    ans = eval(prob)
    user =
int(self.widgets['answer_entry'].get_text())
    self.total += 1
    if ans == user:
        self.correct += 1

self.widgets['result_entry'].set_text('Helyes')
    else:
        self.widgets['result_entry'].set_text(
            'Hibás, a helyes válasz: %d' % ans)
    self.show_results()
```

A C GTK API tartalmazza a `G_CONST_RETURN gchar* gtk_entry_get_text(GtkEntry *entry)` függvényt. Mivel a ***Python*** objektumközpontú nyelv, a kötéseket az osztályhoz tartozó eljárásokként adhatjuk meg. Python kötéseknél a `gtk_` és a `widget_` előtagok kimaradnak a függvénynevek-



2. ábra Math Flash

ből, és az adott kezelőelem Python példányával kerülnek meghívásra. Ugyanez érvényes az összes GTK függvényre is. A `self.widgets` példány használatával a megfelelő *Python* hívás `self.widgets[´problem_entry´].get_text()` formát ölt, ahol a `problem_entry` a *Glade* XML fájlban szereplő beviteli kezelőelem neve.

Ezután a *Python* `eval` függvényével végzem el a feladványra adott válasz kiértékelését. Ez sokkal egyszerűbb, mint saját értelmezőt írni a kifejezés kezelésére. Általános szabály, hogy a szorzás és az osztás elsőbbséget élvez a kivonással és az összeadással szemben. Az `eval` függvény használata biztonsági szempontból aggályos lehet, ha a kiértékelt karakterlánc megadását a felhasználóra bízzuk. Ebben az esetben azonban ezt saját programunk állítja elő, így ezzel kérdéssel nem kell foglalkoznunk.

Ha módosítjuk a *Glade* XML fájlt, futtassuk újra a *GladeGent*, amely elvégzi az új visszahívó eljárások hozzáadását. Eközben az általunk már megírt kódot nem változtatja meg és nem írja felül. Ez alól az `init` metódus az egyetlen kivétel, amit soha ne módosítsunk, mert a *GladeGen* minden futtatásakor újat készít belőle. Az `init` tartalmazza a kezelőelemek és a visszahívók nevét, vagyis muszáj a *Glade* fájl minden frissítésekor újra előállítani. Én például utólag adtam hozzá a statisztika nullázására szolgáló gombot. Amikor újra futtattam a *GladeGent*, akkor hozzáadta az üres `on_reset_button_clicked` eljárást, illetve készített egy új `init` eljárást, amely a `reset_button` kezelőelemet és az `on_reset_button_clicked` visszahívót is tartalmazta. Mivel a kezelőfelület létrehozása futási időben, a *libglade* segítségével történik, további módosításokra nem volt szükség.

Hogyan működik a GladeGen?

A *GladeGen* először megvizsgálja, hogy a megadott modul/fájl létezik-e. Ha nem, létrehoz egy alapszintű fájlt, amelyben a szerző neve és egy osztály szerepel. Ezután feldolgozza a *Glade* XML fájlt, és összeállítja a kezelőelemek és az eseménykezelők listáját. A *Python* az `inspect` modullal teszi lehetővé, hogy egy program meghatározza, egy meglévő *Python* modul milyen függvényeket, osztályokat és eljárásokat tartalmaz, valamint ezekhez mely sorok tartoznak. A *GladeGen* is ezzel vizsgálja meg, hogy mely visszahívókat írtunk már meg, és ezeket nem cseréli le üres visszahívó eljárásra. A program az új visszahívókat az osztályok meghatározásának aljára illeszti. Az `inspect` segítségével a *GladeGen* azt is meg tudja állapítani, hogy mely sorok tartalmazzák az `init` eljárást, majd képes új, a legújabb *Glade* XML fájlban szereplő kezelőelemek és eseménykezelők mindegyikét tartalmazó `init` eljárásra cserélni.

A *Python* az XML fájlok feldolgozására szabványos DOM és SAX felületen keresztül egyaránt képes. A SAX egy eseményvezérelt modell, amelyben a felhasználó adja meg az XML-címkék feldolgozásakor meghívandó függvényeket. A DOM felület a teljes XML fájlt beolvassa a memóriába, majd megfelelő függvényeket biztosít az XML-szerkezet bejárására és az adatok kinyerésére. A *GladeGen* esetében csak az XML fájlban szereplő adatokra van szükségünk, így a DOM felületet egyszerűbb használni. A *Glade* XML fájlok elég kis méretűek, így minden probléma nélkül betölthetők a memóriába, és *Python*-féle megjelenítésük létrehozásával sem kötünk le túl sok memóriát. A DOM felület használatával a *GladeGen* `get_xml` eljárása körülbelül 30 sornyi *Python* kóddal hámozza ki a kezelőelemek és az eseménykezelők nevét a *Glade* XML fájlból.

Összegzés

A *Glade* és a *GladeGen* segítségével gyakorlatilag önműködően oldható meg a grafikus alkalmazások kezelőelemeinek létrehozása. Így megszabadulunk az ezekkel kapcsolatos kód és a visszahívó függvények megírásának unalmas és sokat ismételt feladatától. A két programmal jelentősen felgyorsítható a *Python*/*GNOME*/*GTK* alapú alkalmazások fejlesztése. A példaként elkészített *Math Flash* a 2. ábrán látható. A *GladeGen* program minden a *Python*-t és a *GTK*-t támogató operációs rendszeren, vagyis Linuxon, UNIX-on, Mac OS X-en és Microsoft Windowson egyaránt futtatható.

A rendszer számos további szolgáltatással bővíthető. A létrejövő visszahívó függvények általános `*args` paramétere helyett az átadott értékeket nyíltan, a kezelőelemek és a visszahívó prototípusa alapján is meg lehetne adni. A programot egy a *GladeGenConfig.py* fájlban szereplő beállítások módosítására használható grafikus felülettel is tervezem bővíteni. A *GladeGen* program GPL hatálya alatt jelenik meg. Ha bárki érdeklődne a módosítása vagy bővítése iránt, vegye fel a kapcsolatot a szerzővel.

Köszönetnyilvánítás

Köszönettel tartozom hallgatóim egyikének, Jeremiah Schilensnek, aki a program egy korábbi változatán dolgozott együtt velem.

Linux Journal 2004. július, 123. szám

David Reed az Ohio államban található Columbus városában él feleségével és két kutyájával. 1991 óta dolgozik unixos, 1997 óta pedig linuxos rendszerekkel. Doktori dolgozatát a grafikai térfogatelemzésről írta az Ohio State Universityn. Jelenleg informatikát tanít a Capital Universityn, ahol az informatikai tananyagnak a Python, a C++ és a Java is része. David a dreed@capital.edu címen érhető el.

Adatgyűjtés a Comedi segítségével

Létezik egy szabványos alaprendszer, amely számos különböző adatgyűjtő kártyához egységes API-t biztosít. Aki akarja, akár egy normál számítógép párhuzamos kapuján keresztül is kipróbálhatja.

A legtöbb kutató és mérnök imádja az adatokat. Minél több adatot kapnak, annál szélesebb mosoly ömlik szét az arcukon. Ezeknek az embereknek a mérés a mindene. A folyamatok felismeréséhez, a rendellenes jelenségek felfedezéséhez és a végső következtetések levonásához a laborokban dolgozóknak ügyelniük kell arra, hogy teljes adathalmazokat gyűjtsenek.

Az adatgyűjtés fogalma számos ötletet és fogalmat ölel fel. A legtöbb kutató és mérnök ugyanakkor egyetért abban, hogy az adatgyűjtés valamilyen természetes folyamat jellemzőinek mérését jelenti. Lehet szó egyszerű hőmérsékletmérésről, vagy akár olvadt acél szennyeződéseinek vizsgálatáról.

A számítógépek világában az adatgyűjtés általában valamilyen feszültség mérésével történik. Ilyenkor rendelkeznünk kell valamilyen érzékelővel vagy műszerrel, amely a számítógép számára mérhető feszültségeket állít elő. Fontos persze az is, hogy ismerjük a mért jellemző és az érzékelő feszültségkimenete közötti összefüggést. Jó esetben a kapcsolat lineáris. Ilyen például az, ha egy foknyi hőmérsékletkülönbséghez mindig 0,1 voltos feszültségeltérés tartozik.

A korszerű alaplapokon beépített érzékelők találhatók, amelyek képesek a teljes rendszer állapotáról képet alkotni. Ilyen eszköz például a ***National Semiconductor LM78*** jelű terméke. Ezek az érzékelők a gyakorlatban általában a hűtőventilátorok fordulatszámát, a processzormag feszültségét és hőmérsékletét, illetve a merevlemezek fordulatszámát mérik. Az adatokat maga a lapka gyűjti össze, a processzorhoz pedig egy soros buszon keresztül juttat el őket. A nyílt forrású ***lm_sensors*** projekt (➲ ***secure.netroedge.com/~lm78***) programja az alaplapok működésének figyelését számos szempont szerint teszi lehetővé.

A normál személyi számítógépek ettől függetlenül nem rendelkeznek általános, analóg adatok begyűjtésére alkalmas csatolófelülettel. Így ha külső feszültségeket akarunk mérni velük, akkor új felületre van szükségünk. Pontosan ilyen célra tervezik a ***PCI*** és ***ISA*** buszos kivitelben egyaránt létező adatgyűjtő (data acquisition, DAQ) kártyákat, amelyek számos gyártó termékkatalógusaiban megtalálhatók.

1. táblázat *Gyakoribb adatgyűjtő csatorna típusok*

Név	Leírás
Analóg bemenet	Külső jelek, például feszültségek mérése
Analóg kimenet	Változó jel küldése
Digitális be/kimenet	Diszkrét be/ki jel küldése, általában 0 volt a kikapcsolt, 5 volt a bekapcsolt állapot
Számláló	Impulzusok számlálására vagy frekvencia mérésére alkalmas
Időzítő	Két digitális impulzus beérkezése közötti időt méri

A Comedi projekt

A legtöbb Linux-felhasználó alighanem jól tudja, miféle gondokkal jár akár csak egy-egy nyomtató kapcsán a gyártók, a márkák, a típusok és az illesztőprogramok követése. Általános jelenség, hogy bármilyen szabványosítási törekvés csakhamar hatalmas fejlesztési projektté növi ki magát. Ha a tervezet kellő támogatást kap, akkor szabvány születik belőle. Vannak gyártók – ilyen például a ***National Instruments*** – amelyek eleve adtak ki linuxos illesztőprogramokat adatgyűjtő eszközeikhez, és vannak, amelyek nem.

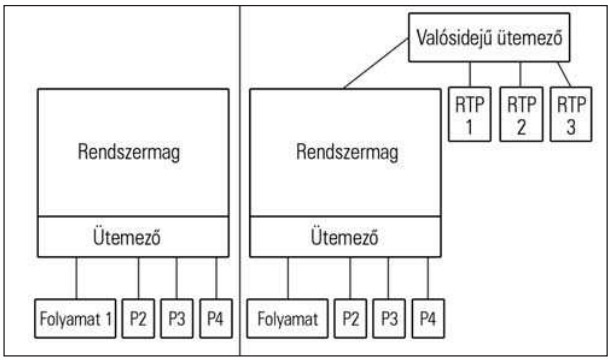
A ***Comedi***, vagyis a ***Control and Measurement Device Interface*** (vezérlő- és mérőeszköz felület) a linuxos adatgyűjtő illesztőprogramok és könyvtárak szabványos készlete. Fejlesztését 1996-ban ***David Schleef***, kezdte meg.

A ***Comedi*** célja több kártyagyártó és -modell támogatása egyetlen közös felületen keresztül. Az API tervezésekor lényegében a modularitás és a bonyolultság között kellett egyensúlyt teremteni. A többi linuxos illesztőprogram fejlesztéséhez hasonlóan a munka egy részéhez rengeteget kell búnyi a különféle eszközök leírásait, más részéhez visszafejtéseket kell végezni, bár természetesen a gyártók által a ***Comedi*** fejlesztéséhez a saját termékükre kiterjedően nyújtott támogatást sem szabad elfeledni.

Hogyan működik?

A ***Comedi*** két részből áll. Maga a ***Comedi*** illesztőprogramok gyűjteménye, amelyeket a rendszermag térbe kell betölteni, a ***comedilib*** pedig ezekhez az illesztőprogramokhoz nyújt

1. kép Légáramm mérő eszköz



2. ábra A normál és a valós idejű linuxos folyamatütemezés összehasonlítása

hozzáférést a felhasználói térből. A ***Comedi*** átlátszósága a ***comedilib*** szolgáltatásainak köszönhető. ***Comedire*** épülő programokat *C* vagy *C++* nyelven lehet írni, ezen kívül ***Perl*** és ***Python*** kötések is léteznek hozzá.
A ***Comedi*** mindent csatornákra, aleszközökre és eszközökre oszt fel. A csatorna a legalacsonyabb mérési vagy vezérlési szint. Több azonos típusú csatorna együtt egy aleszközt alkot, a teljes eszköz pedig ilyen aleszközökből áll össze.
A ***Comedi*** használatakor először egy ***Comedi*** illesztőprogram töltődik be a memóriába. Ezután a `/usr/sbin/comedi_config` fut le, amely létrehozza az illesztőprogram kötését a megfelelő ***Comedi*** eszközhöz, ez lehet például a */dev/comedi0*. Végül a DAQ-kártya által biztosított szolgáltatásokat a ***comedilib*** függvényei révén lehet elérni.

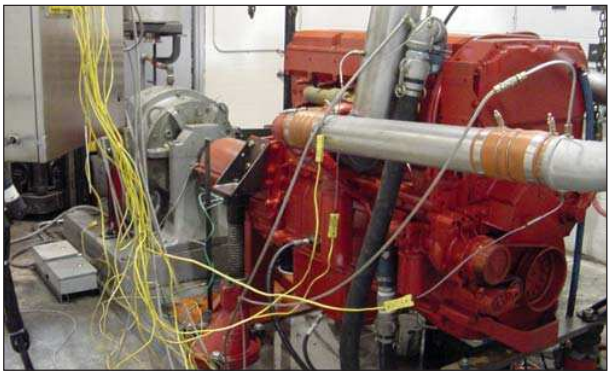
Laboratóriumi példa

DAQ és a ***Comedi*** együttes használatára többek közt az ***Analytical Engineering, Inc. (AEI)*** aerodinamikai laboratóriumában láthatunk példát. Itt egy ventilátor légáramát különféle méretű nyílásokon vezetik keresztül, a technikusok pedig egy egyedi program segítségével követni tudják a nyílásokban létrejövő nyomás alakulását. A mérés alapján ki lehet számítani a nyílásokon átáramló légtömeg nagyságát. Ezek a mérések és számítások alapvető jelentőséggel bírnak, mivel a technikusok ezek alapján tudják ellenőrizni, hogy a különféle műszerek kalibrálása helyes-e.

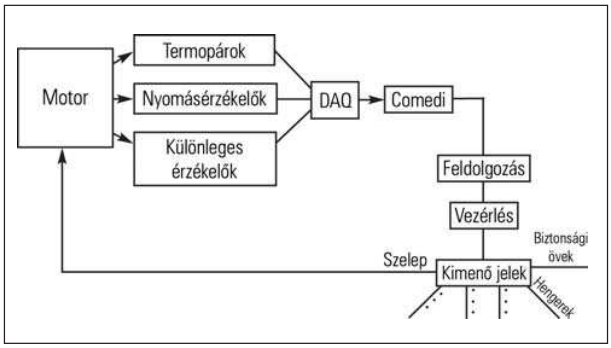
1. kódrészlet Példaprogram feszültségszint lekérdezésére az 1-es csatornáról

```
#include <stdio.h>
#include <comedilib.h>
const char *filename = "/dev/comedi0";
int main(int argc, char *argv[])
{
  lsampl_t data;
  int ret;
  comedi_t *device;
  /* A kártya melyik eszközét akarjuk
     használni? */
  int subdevice = 0;
  /* A csatorna kiválasztása */
  int channel   = 0;
  /* Az elérhető tartományok egyikének
     kiválasztása */
  int range     = 0;
  /* Mérések végzése földhivatkozással */
  int analogref = AREF_GROUND;
  device = comedi_open(filename);
  if(!device){
    /* Nem tudtuk megnyitni az eszközt - hiba
       történt */
    comedi_perror(filename);
    exit(0);
  }
  /* Adat beolvasása */
  ret=comedi_data_read(device,subdevice,
    channel,range,analogref,&data);
  if(ret<0){
    /* Valamilyen hiba történt */
    comedi_perror(filename);
    exit(0);
  }
  printf("A következő adatot kaptuk: %d\n",
         data);
  return 0;
}
```

A tényleges tömegáramlást még nehezebb pontosan kiszámítani. Meghatározásához kétféle légnyomást, három légáram hőmérsékletet, a nedvességtartalmat, a légköri nyomást és a magasságot kell ismerni. Mindezeket a mennyiségeket hétköznapi kereskedelemben kapható eszközökkel is át lehet alakítani feszültségekké. Az egyik legnépszerűbb csatolófelület az ***5B***. Ilyen típusú moduláris blokkokat használva mindezeket a mennyiségeket a DAQ kártya által olvasható feszültségekké lehet formálni. A ***Comedi*** segítségével mindezeket a feszültségeket könnyedén, a `comedi_data_read` függvény meghívásával tudjuk beolvasni. Ez a kiválasztott csatornát figyelve egyszerűen egy értéket ad vissza, amely lehet például 3421. De vajon mit jelent ez a szám?
A DAQ kártyák meghatározott felbontással dolgoznak. A leggyakoribb a 12 bites pontosság. Mindegyikhez egy adott feszültségtartományban dolgozik, amelyre a mérési eredményeket leképezi. Mivel 12 biten a 0-4095 értéktarto-



3. kép Motor vizsgálat közben



4. ábra A motorok vezérlése és jellemzőinek Comedi segítségével végzett mérése

mány ábrázolható, könnyen kiszámíthatjuk, hogy a 3421 jelentése: 3421/4095 *(a teljes mérési tartomány). Ha például eszközünk a [0, 5] feszültségtartományban dolgozik, akkor a 3421 értéknek 4,177 voltos feszültség felel meg. Mindezen adatok birtokában, illetve tudva, hogy az 5B blokk a [0 volt, 5 volt] feszültségtartományra a [0 °C – 100 °C] hőmérséklettartományt képezi le, néhány pillanat alatt ki tudjuk számítani, hogy a fenti feszültség 83,56 °C-os hőmérsékletet jelez. Gyúrjuk össze a szükséges méréseket, csapjunk hozzá egy csinos grafikus felületet, és ismételjük meg a mérést minden másodpercben.

Természetesen ennél bonyolultabb adatgyűjtéseket is végezhetünk. Az adatok begyűjtésekor fontos szempont a sebesség. Elég gyorsnak kell lennünk ahhoz, hogy a minták között semmilyen fontos adatot ne hagyjunk figyelmen kívül. Ennek elérésére a ***Comedi*** egy parancsfelületet is biztosít, amellyel szinkronizált mintavételt lehet végezni.

A DAQ-kártya képességeitől függően az időzítésről program alapú és a kártya által biztosított megszakítások egyaránt gondoskodhatnak.

A ***Comedi*** a legtöbb adatgyűjtési feladatot kiválóan képes ellátni. Lényegében a ***Comedi*** szolgáltatásait csak a futtatására használt hardver képességei korlátozzák. Az olcsóbb kártyák általában kisebb mintavételi gyakorisággal dolgoznak. Ha gyorsabb adatgyűjtést akarunk végezni, akkor drágább kártyát kell választanunk. Ezek DMA-kezelésre is képesek, az adatgyűjtést pedig saját processzoruk vezérli. A ***Comedi*** ilyenkor csak a pufferelt adatok továbbítását irányítja.

A gyors beolvasás ugyanakkor nem jelent egyben gyors feldolgozást is. A közönséges ***Linux*** rendszermag viselkedése időzítési szempontból kiszámíthatatlan, ezért az adatgyűjtés mellett valós idejű feldolgozást is végezni gyakorlatilag lehetetlen. Ehhez a folyamatok ütemezését szigorú szabályok alapján kellene végezni. Természetesen erre is van megoldás. A ***Linux Real-Time Application Interface*** (***RTAI***, valósidejű alkalmazásfelület) és az ***RTLinux*** csupán két példa a számos lehetőség közül, amelyekkel feljavítható a rendszermag időzítésvezérlése. Mindkét csomaghoz létezik ***Comedi*** felület. A valós idejű felületek mögötti alapötlet roppant egyszerű. A rendszermagot nem monolitikus folyamatként kell futtatni, hanem egy kis méretű és hatékony ütemező gyermekfolyamataként. Így a rendszermag nem tudja blokkolni a megszakításokat, illetve szükség esetén el lehet venni tőle az erőforrásokat. Ezután bármely a rendszer felett valós idejű vezérlési lehetőséget igénylő alkalmazás bejegyezheti magát az ütemezőnél, és olyan gyakran foszthatja meg a rendszermagot az erőforrásoktól, amilyen gyakran csak szeretné.

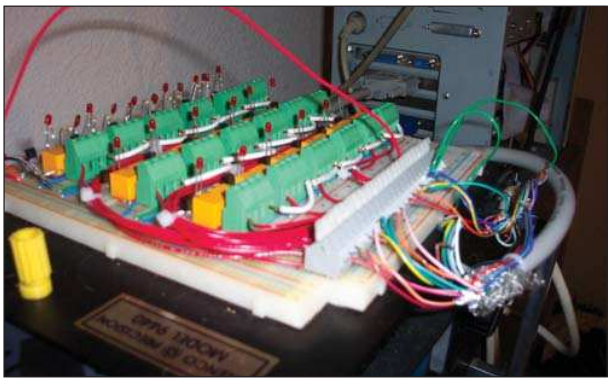
Vissza a laboratóriumi példához

Az ***AEI*** több tesztkamrát is fenntart dízelmotorok teszteléséhez. Minden cellában egy motor található, számos hőmérséklet- és nyomásmérő műszerrel felszerelve. Be szoktak építeni egy frekvenciamérőt is, amely a motor fordulatszámát figyeli. Végül a motort egy erőmérőhöz csatlakoztatják, amely valós vezetési helyzeteket szimulál úgy, hogy változó ellenállást fejt ki a működő motorral szemben. A létrejövő nyomatékot szintén mérjük.

A motoradatok beolvasási sebessége viszonylag kicsi, másodpercenként mindössze 20 mintát veszünk. Ha az adatok begyűjtése volna az egyetlen feladat, akkor az egész rendszer meglehetősen egyszerű lenne. Csakhogy az újabb adatkészletek beolvasásakor különféle beállításokat kell finomhangolni és szabályozni. A motor sebességét adott értéken kell tartani, ehhez változtatni kell a fojtószelep állását, illetve az erőmérő által adott terhelést. A cellának a hűtőfolyadék áramlását szabályozó szelepeit ugyancsak folyamatosan állítani kell, mert a hűtőfolyadék hőmérsékletét állandó értéken kell tartani. Biztonsági méréseket ugyancsak kell végezni, ezekkel ellenőrizhető, hogy katasztrófához vezető körülmények nem alakultak-e ki.

Ráadásul minden ellenőrzést és az új értékek beállítását még az előtt kell elvégezni, hogy egyéb teendői elvégzésére a rendszermag átvenné a vezérlést. Ha a ***Linux*** rendszermag maga gondoskodna erről az ütemezésről, nagy valószínűséggel minden megfelelően működne. Sajnos azonban nem lehet előre megmondani, hogy a teljes folyamat egyes lépései mikor kerülnek végrehajtásra. Az említett valós idejű kiterjesztések használatával ugyanakkor ez a probléma automatikusan megoldódik.

A valós idejű rendszermagoknak hátrányaik is vannak. Amikor a valós idejű ütemező meghatározott időszeletben futtat egy folyamatot, a ***Linux*** rendszermag futása lényegében megakad. A valós idejű folyamatnak ezért gyorsnak és hatékonynak kell lennie, és a lehető leghamarabb vissza kell adnia a vezérlést a rendszermagnak. Ha ezt elmulasztja, akkor a rendszer egyéb, nem valós idejű részei eléggé akadozva fognak futni. Ha pedig a valós idejű folyamatban valami hiba történik, és a rendszermag soha nem kapja vissza a vezérlést, akkor a teljes rendszer összeomolhat.



5. kép Csatoló áramkör a lámpák párhuzamos kapuról végzett vezérléséhez

Gyakorlati példa

A laboratóriumok mellett a ***Comedi*** akár otthon is használható. Egy egyszerűbb adatgyűjtő kártyához már 20-60 ezer forint körüli áron hozzájuthatunk, a gyártó nevétől, a kártya bonyolultságától és mintavételi gyakoriságtól függően. Egy ilyen eszközzel lakásunk különféle pontjain mérhetjük a hőmérsékletet, vagy mágneses érzékelővel figyelhetjük, hogy nem felejtettük-e nyitva a garázsajtót.

A személyi számítógépek érdekes jellegzetessége, hogy a párhuzamos kapuk vonalait külön is lehet vezérelni. A ***Comedi*** segítségével ezeket a vonalakat roppant egyszerűen tudjuk be- és kikapcsolni. Megfelelő relével párosítva pedig ezekkel gyakorlatilag bármit ki-be tudunk kapcsolgatni. Bár a párhuzamos kapuk az említett 0-5 voltos feszültségszintekkel dolgoznak, kellő áramerősséget nem képesek leadni. Párhuzamos kaput tehát nem szabad közvetlenül a vezérelni kívánt készülékhez csatlakoztatni, mindig valamilyen köztes áramkört kell beiktatni. Ilyen áramkörök készítéséhez számos weboldalon találhatunk leírásokat.

Én egy öreg 486-oson futtatom a ***Comedit***, és két párhuzamos kapu segítségével hozom létre az évi nyaralásunkkor szokásos fényjátékot. A házban a lámpák a szokásos módon vannak felszerelve, de mindegyikhez külön érpár vezet, amelyek a vezérlőszobában futnak össze (esetemben ez egy üresen álló hálószoba). Az érpárok egyedi áramköri laphoz csatlakoznak. Ezen találhatók azok a mechanikus relék, amelyek áramot adnak a lámpáknak, ha 5 voltos jelet kapnak a párhuzamos kapuról. Egy egyszerű C programot használok, amely ***Comedi*** függvényhívásokkal egyenként vezérli a párhuzamos kapu vonalait, vagyis be- és kikapcsolja a világítótesteket. Azt, hogy az egyes lámpákat mikor kell be- és kikapcsolni, egy egyszerű szöveges fájlban tudom megadni. A szomszédom nagy örömére...

Összefoglalás

Az adatgyűjtés lehetősége a laboratóriumokban rendkívül értékes. A ***Comedi*** által biztosított általános felülettel a ***Linux*** a legkülönfélébb DAQ-kártyákkal is kiválóan használható. Ahogy a ***Linux*** egyre népszerűbbé válik, úgy a ***Comedihez*** hasonló felületek fontossága is növekedni fog.

Az egyszerű DAQ-kártyák lassanként megfizethetővé válnak, a ***Linux*** alapú adatgyűjtés pedig valószínűleg egyre inkább elnyeri majd a területtel csak hobbiként vagy barkácsolási céllal foglalkozók tetszését. Ami korábban költséges hardver és programok vásárlását igényelte, az ma már – a legkülönfélébb területeken – bárki számára hozzáférhető.

2. kódrészlet Parancsok és ravaszok segítségével végzett, az előbbinél bonyolultabb pásztázást szemléltető kód

```
/* Cél: A Comedi beállítása kétcsatornás adat-
gyűjtésre és mindkét adatkészlet kétszeri beol-
vasása. Az adatgyűjtést egy digitális vonalon
érkező ravaszjel hatására indítjuk el.
  */
  comedi_cmd c, *cmd=&c;
  unsigned int chanlist[2];
  /* A CR_PACK egy különleges Comedi makró,
     amely a csatorna, a tartomány és
     a földhivatkozás beállítására szolgál.
  */
  chanlist[0] = CR_PACK(0,0,0);
  chanlist[1] = CR_PACK(1,0,0);
  /* Melyik aleszközt kell használnunk? */
  /* A legtöbb kártyán a nullás aleszköz egy
     analóg bemenet. */
  cmd->subdev        = 0;
  cmd->chanlist      = chanlist;
  cmd->chanlist_len = n_chan;
  /*A parancs elindítása a külső digitális vo-
    nalon érkező jelzés hatására. A start_arg ál-
    tal megadott digitális csatornát használjuk.
  */
  cmd->start_src = TRIG_EXT;
  cmd->start_arg = 3;
  /* A ravaszjel után azonnal megkezdjük a pász
     tázást. */
  cmd->scan_begin_src = TRIG_FOLLOW;
  cmd->scan_begin_arg = 0;
  /* A pásztázás után azonnal elvégezzük az
     átalakítást. */
  cmd->convert_src = TRIG_NOW;
  /* A scan_end_arg csatorna lekérdezése után
     befejezzük a pásztázást.
  */
  cmd->scan_end_src =      TRIG_COUNT;
  cmd->scan_end_arg =      2;
  /* stop_arg számú pásztázás után a parancsot
     leállítjuk. */
  cmd->stop_src =          TRIG_COUNT;
  cmd->stop_arg =          2;
  /* A parancs indítása */
  comedi_cmd(device, cmd);
```

Linux Journal 2004. augusztus, 124. szám

Caleb Tennis 1996 óta használ Linuxot. Ő volt a KDevelop Project egyik vezetője, jelenleg elsősorban a KDE Gentoo alatti változatának karbantartásával foglalkozik. Egy dízelmotor-tesztlabor mérnöki munkáit felügyeli, s eközben részidőben egy helyi főiskolán linuxos témákban oktat.

Fájlrendszer-címkézés a SELinuxban

A SELinux sokkal kifinomultabban vezéreli a fájlokhoz való hozzáférést. Lássuk!

Most, hogy az ***NSA Security Enhanced Linux*** (***Fokozott Biztonságú Linux***) része lett a 2.6-os rendszermagnak, és így eljut a különböző ***Linux*** terjesztésekbe, valószínűleg egyre többen telepítik majd a ***SELinuxot***, és kezdenek kísérletezni vele. Ezeknek a jövőbeni felhasználóknak kíván segítséget nyújtani ez a cikk, közelebbről is megvizsgálja a SELinux alatti fájlrendszer-címkézést. Bár ez az írás alapvetően a középhaladóknak szól, a következő bekezdésben rövid áttekintést adok a SELinux néhány alapelvéről, a hálózaton található forrásokban pedig szerepel néhány hivatkozás a részletesebb információk forrásaira. ***Faye Coker*** bemutató cikke (***Linux Journal, 2003. augusztus***) különösen ajánlott azoknak, akik csak most kezdenek ismerkedni ezzel a különleges területtel.

SELinux áttekintés: címkézés és hozzáférés-szabályozás

A ***SELinuxban*** valamennyi, biztonsági szempontból fontos objektumhoz, például a feladatokhoz, a fájlleírókhoz és magukhoz a fájlokhoz egy-egy biztonsági környezet tartozik, vagyis egy-egy olyan címke, amely magában foglalja az adott objektumhoz kapcsolódó biztonsági jellemzőket. A szabványos ***SELinuxban*** ez egy kettősponttal elválasztott karakterlánc, amely azonosító, szerep és típus értékekből áll. Ezeket a címkéket egy ***biztonsági kiszolgáló*** nevű rendszermag-összetevő rendeli hozzá a megfelelő objektumokhoz, mégpedig a biztonsági házirend adatbázisában megadott szabályok alapján. A szerző munkaállomásán található */etc/shadow* fájl biztonsági környezeti címkéje például a következő:

```
system_u:object_r:shadow_t
```

A `system_u` és az `object_r` a fájlokban használt általános azonosító és szerep értékek, míg a `shadow_t` a fájl típusa. Ez utóbbi egy olyan jellemző, amely meghatározza, hogyan lehet elérni az adott fájlt. Egy folyamatot a következőképpen lehet címkézni:

```
root:staff_r:staff_t
```

A ***SELinux*** azonosító itt `root`, amelyet a ***SELinux*** a szabványos ***UNIX*** azonosítónál tartósabb azonosítófajtaként rendel hozzá. A szerep itt `staff_r`, ami azt jelöli, hogy a folyamat az ehhez a szerephez rendelt összes jogosultsággal rendelkezik. A folyamathoz kapcsolt típus a `staff_t`. A folyamatoknál a típus jellemző meghatározza, hogy azok hogyan férhetnek hozzá az objektumokhoz, és hogyan léphetnek kapcsolatba más objektumokkal. A folyamatok típus jellemzőjét gyakran tartományként is emlegetik.

Hozzáférés-szabályozási döntések

A ***SELinux*** a biztonsági környezeti címkék segítségével a folyamatok és az objektumok közötti kapcsolatrendszert határozza meg. A rendszer működése közben folyamatosan döntéseket hoz azzal kapcsolatban, hogy melyik folyamat melyik objektumhoz férhet hozzá, és pontosan hogyan. De hogyan történik a gyakorlatban mindez? A ***SELinux*** „kapaszkodókat" (hooks) helyez el az alapvető rendszermagkód stratégiai pontjain, például ott, ahol a felhasználó éppen olvasni kezdi a fájlt. Ezek a kapaszkodók lehetővé teszik a ***SELinux*** számára, hogy felfüggessze a rendszermag rendes adatforgalmát, és kifinomult döntéseket kényszeríthessen ki a hozzáférések szabályozása terén a megfelelő kérésekkel. A hozzáférés-szabályozási döntések általában folyamatok (például `cat`) és objektumok (például */etc/shadow*) között történnek, meghatározott jogosultságok megszerzése végett. (Példánknál maradva a `cat`-nek nyilván olvasási jogosultságot kellene szereznie az adott fájlra.). A döntéskérések a hozzáférés vektortárba (***Access Vector Cache*** – ***AVC***) kerülnek, amely átadja a kéréseket a biztonsági kiszolgálónak, kiértékelésre. A biztonsági kiszolgáló a kapott adatokat egyezteti a biztonsági házirend adatbázis tartalmával, majd meghatároz egy eredményt, amely az ***AVC***-ben tárolódik, és visszatér a megfelelő ***SELinux*** kapaszkodóhoz. A ***SELinux*** kapaszkodó ezután vagy engedélyezi a forgalom folytatását, vagy egy `EACCES` értéket ad vissza, a döntés eredményétől függően. A folyamatokhoz és az objektumokhoz rendelt biztonsági környezeti címkék ezeknek a döntéseknek a meghozatalát támogatják. Az *1. ábrán* ennek a folyamatnak egy egyszerűsített változatát láthatjuk.
A következő kód azt szemlélteti, hogy a biztonsági környezeti címkék hogyan működnek egy valódi rendszeren, illetve mit lát maga a felhasználó az egész folyamatból:

```
$ id -Z
root:staff_r:staff_t

$ cat /etc/shadow
cat: /etc/shadow: Permission denied
```

```
Az ellenőrzési napló a következőt rögzíti:
avc:  denied  { read } for  pid=13653 exe=/bin/cat
name=shadow dev=hda6 ino=1361441
scontext=root:staff_r:staff_t
tcontext=system_u:object_r:shadow_t tclass=file
```

A `cat` program, amelynek a biztonsági környezeti címkéje esetünkben `root:staff_r:staff_t`, nem kapta meg az engedélyt egy `system_u:object_r:shadow_t` címkéjű fájl olvasására.

A ***SELinuxnak*** természetesen halvány fogalma sincs arról, hogy mi az a `cat` vagy mit tartalmaz a ***/etc/shadow*** fájl, őt egyszerűen csak a megfelelő biztonsági környezeti címkék, a célobjektum (ebben az esetben a fájl) osztálya és az éppen kérelmezett engedély típusa érdeklik. Az ezekkel megadott információ számára tökéletesen elegendő a döntés meghozatalához.

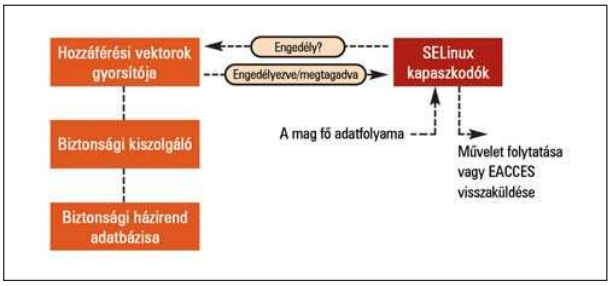
A ***SELinux*** tervezésében fontos szempont, hogy a címkék az objektumok összes biztonsági jellemzőit magukban foglalják. A rendszermagban a biztonsági kiszolgáló, a felhasználói térben pedig a ***libselinux*** értékeli ki azokat a megfelelő pillanatokban. Maga a rendszermagkód és a felhasználói tér többi része semmi egyebet nem tesz, csupán számára ismeretlen adatként továbbadja a címkéket. Ennek a rendszernek az egyik óriási előnye az, hogy a címkéket anélkül lehet újabb biztonsági jellemzőkkel kiegészíteni, hogy ehhez újra kellene fordítani az alkalmazásokat vagy újratervezni az alapvető ***SELinux*** kódot.

Kiterjesztett jellemzők

Egy jellegzetes lemez alapú ***Linux*** fájlrendszerben minden fájlt egyedileg azonosít egy olyan fájlleíró, amely a fájl kulcsfontosságú adatait tartalmazza, beleértve a ***UNIX*** tulajdonjogot és a hozzáférési adatokat. Amikor a rendszermag egy fájlra hivatkozik, akkor a lemezről a memóriába olvassa ezt a fájlleírót. A ***UNIX*** szabványos jogosultságellenőrzése egyszerűen a fájlleíróban szereplő adatokat használja. A ***SELinux*** tulajdonképpen ezt a szabványos ***UNIX*** biztonsági protokollt terjeszti ki, és biztonsági környezeti címkéket használ a finomított hozzáférés-szabályozási döntések meghozatalára.

Maga a ***Linux*** megvalósítja a kiterjesztett jellemzőket (***Extended Attributes*** – ***EAs***), más néven ***EAs*** vagy ***xattrs***. Ezek a név/érték párok fájlokhoz rendeltek, akár a kiterjesztések a hagyományos fájlleíró alapú jellemzőkhöz. A kiterjesztett jellemzők segítségével úgy lehet szabványosított módon szolgáltatásokat adni a fájlrendszerekhez, hogy a jellemzők felületei fájlrendszerektől függetlenek legyenek. A kiterjesztett jellemzőkkel kapcsolatos szolgáltatások közé tartoznak például a hozzáférés-szabályozási listák (***Access Control Lists*** – ***ACL***), a karakterkészletleíró-adatok tárolása a fájladatok mellett és a ***SELinux*** biztonsági környezet címkézés.

A kiterjesztett jellemzők a névtereken belül tárolódnak, lehetővé téve ezzel a különböző osztályokba tartozó kiterjesztett jellemzők elkülönített kezelését. A hozzáférésszabályozási listák a `system.posix_acl_access` és a `system.posix_acl_default` névterekben kapnak helyet. A ***SELinux*** biztonsági környezeti címkéi a `security.selinux namespace` névtérben találhatóak.



1. ábra Az SELinux hozzáférés-szabályozási döntései a rendszermagban

Akit részletesebben is érdekelnek a ***Linux*** kiterjesztett biztonsági jellemzői az olvassa el az `attr(5)` súgóoldalt.

Kiterjesztett jellemzők biztonsági címkézése

A kiterjesztett jellemzők természetesen „kézzel” is módosíthatók a `getfattr(1)` és a `setfattr(1)` segédprogramokkal. Ha például látni akarjuk egy fájl ***SELinux*** biztonsági környezeti címkéjét, akkor a következőt kell tennünk:

```
$ getfattr -n security.selinux /tmp/foo
getfattr: Removing leading ´/´ from absolute
➥path names
# file: tmp/foo
security.selinux="root:object_r:sysadm_tmp_t\000"
```

Figyeljük meg a kiterjesztett jellemző biztonsági névterének leírását! A rendszer használatát megkönnyítendő a ***SELinux***-hoz kapunk egy `getfilecon(1)` nevű burkolóprogramot is, ami bizonyos helyzetekben nagyon megkönnyítheti az életet. Segítségével ugyanis nem nekünk kell meghatározni a kiterjesztett jellemző névteret, és tisztább eredményt is ad.

A szöveg alapú címkék használata biztosítja, hogy értelmes, olvasható biztonsági jellemzők tárolódjanak a fájladatokkal. Ezek a címkék változatlanok maradhatnak vagy lefordíthatók, ha a fájlrendszer egy másik, valószínűleg más biztonsági rendet alkalmazó rendszerre fűződik be. Ennek ellenpéldája, ahogy a fájl tulajdonosa felhasználói számazonosítóként tárolódik a fájl leírójában. A felhasználói azonosító jellemzően egy értelmes értékhez van hozzárendelve az ***/etc/passwd*** fájlon keresztül – egy másik rendszerben más jelentése lehet.

Ahhoz, hogy egy fájlrendszer támogassa a ***SELinux*** biztonsági környezet címkéket, kiterjesztett jellemző támogatásra és kiterjesztett jellemző biztonsági névtér kezelőre van szüksége. Jelenleg az ilyen fájlrendszerek közé tartozik az ***ext3***, az ***ext2***, az ***XFS*** és a ***ReiserFS*** – ez utóbbi külső foltot alkalmaz. Ráadásul a ***devpts*** fájlrendszer hamis biztonsági kezelővel rendelkezik, amely megengedi a kiterjesztett jellemző alapú hozzáférést a ***pty***-k rendszermagban található címkéihez.

Tehát, mikor kerülnek címkézésre a fájlok? A ***SELinux*** rendszer telepítése során, a `setfiles(8)` segédprogram jellemzően arra szolgál, hogy a fájlrendszer összes olyan fájlját megcímkézze, amely támogatja a kiterjesztett jellemző biztonsági címkézést. Az olyan csomagkezelő eszközök, mint az ***RPM***, szintén megcímkézhetik a fájlokat a telepítés so-

A SELinux fájlrendszer-címkézés fejlődése

A ***SELinux*** első változata 2000-ben más elvet használt a fájlrendszerek címkézésére, mint amit az ebben a cikkben tárgyalt kiterjesztett jellemzők megközelítés használ. Az állandó biztonsági azonosítók (***Persistent Security IDs*** – ***PSIDs***) a biztonsági környezet címkék egész megvalósításai az ext2 fájlleíró használaton kívüli mezőjében tárolódtak. A fájlok leképezését mindkét fájlrendszerben arra használta a ***SELinux***, hogy megkeresse a fájlok ***PSID***-jét a fájlleíró alapján, és leképezze a biztonsági környezet címkére. Ennek a megközelítésnek az volt a hátránya, hogy minden fájlrendszert külön módosítani kellett, hogy támogassa a ***PSID***-ket. Ezért a kiterjesztett biztonságra nem volt jó általános megoldás az árral szemben haladó rendszermagban. Az ***LSM*** projekttel egy általánosított hozzáférés-szabályozási váz valósult meg a ***Linux*** rendszermagban. Mivel az ***LSM***-ben nincsenek fájlrendszerre jellemző kapaszkodók, a ***SELinux*** eltávolodott a módosított fájlrendszer megközelítéstől, és a ***PSID***-ket hagyományos fájlban tárolta, a leképező fájlok mellett. Ez lehetővé tette a ***SELinux*** számára, hogy tisztán ***LSM*** alkalmazásként legyen használható, rendszermag-foltozás nélkül. Ahhoz is hozzájárult, hogy a címkézés több fájlrendszeren működjön, de teljesítmény és következetesség szempontjából nem volt optimális. A fájlok rendszermagból való elérése továbbra is gondot okoz. A ***SELinux*** fősodorbeli rendszermagba olvadásának folyamataként, a közösségtől kapott több visszajelzés alapján a ***SELinux*** a jelenlegi kiterjesztett jellemzőkre épülő fájlrendszer-címkéző modellre váltott. A kiterjesztett jellemzők szabványos alkalmazásprogramozói felülettel rendelkező alkalmazásokat biztosítanak, a fájlrendszereket pedig hasonlóképpen más biztonsági modellek is használhatják, még ugyanazon a fájlrendszeren belül is, a kiterjesztett jellemző névterek által biztosított elkülönítés alkalmazásával.

rán, a rendszergazdáknak viszont gyakran kézzel kell beállítaniuk a biztonsági környezetet, a `chcon(1)` vagy a `setfilecon(1)` segédprogrammal.

Fájlok létrehozása

Amikor létrejön egy fájl, a biztonsági rend egy megfelelő szabálya általában leírja, hogyan kell címkét hozzárendelni a szülőkönyvtár és az aktuális feladat biztonsági környezete alapján. Íme egy példa:

```
$ id -Z
root:staff_r:staff_t
$ ls -dZ /tmp
drwxrwxrwt+ root  root system_u:object_r:tmp_t
/tmp
$ touch /tmp/hello
$ getfilecon /tmp/hello
/tmp/hello       root:object_r:staff_tmp_t
```

Ebben az esetben a biztonsági rend egy olyan szabályt tartalmaz, amely megszabja, hogy a `staff_t` által a `tmp_t` könyvtárban létrehozott fájlokat a `staff_tmp_t` típus címkével kell ellátni. Ha nincs egyértelmű szabály, akkor a fájlokat a szülőkönyvtár környezetével kell megcímkézni.
A kitüntetett alkalmazások felülbírálhatják az imént említett szabályt, úgy, hogy biztonsági környezetet írnak a */proc/self/attr/fscreate* fájlhoz. Ezután ez a biztonsági környezet szolgál az összes újonnan létrehozott fájl megcímkézésére. A `setfscreation(3)` könyvtárfüggvény magában foglalja ezt a lehetőséget. Ahhoz, hogy kézzel visszaállítsunk egy biztonsági környezetet, használjuk a `restorecon(8)`-t.
Az olyan helyzetek kezelésére, amikor címke nélküli fájlok keletkeznek, fejlesztés alatt áll egy `fsck`-szerű segédprogram. Ez az indításkor futtatandó alkalmazás biztosítja majd, hogy minden fájl helyesen címkézett, mielőtt több felhasználós üzemmódba lép.

Viselkedésmódok címkézése

Az előző részben az olyan fájlrendszerekben való fájlcímkézésről volt szó, amelyek támogatják a lemezen található kiterjesztett jellemzőket, és rendelkeznek kiterjesztett jellemző biztonsági névtér kezelővel. Amikor egy ilyen fájlrendszer helyesen van befűzve, akkor úgy mondják, hogy a `xattr` címkéző viselkedésmódot alkalmazza.
Amikor a ***SELinux*** kezdeti értéket ad egy fájlrendszernek, például a befűzés során, a következő naplóüzenet keletkezik:

```
SELinux: initialized (dev hda6, type ext3), uses
xattr
```

A `uses xattr` záradék azt jelenti, hogy a fájlrendszer a fent leírt `xattr` címkéző viselkedésmódot alkalmazza. Sok fájlrendszer nem támogatja a kiterjesztett jellemzőket, és azok közül, amelyek támogatják, sem mind rendelkezik biztonsági névtér kezelőkkel. A lemezes fájlrendszereknél lehet, hogy még senki sem végezte el a kódolási munkát, vagy egyszerűen a kiterjesztett jellemzőknek nincs értelme a *vfat*-hez hasonló öröklött fájlrendszereken.
A pszeudo-fájlrendszerek burjánzása a Linux alatt kezdődött. A fájlrendszerek egyre kedveltebb felhasználó-rendszermag alkalmazásprogramozói felületté válnak. Ezek közül a ***procfs*** a legszembetűnőbb, amely a felhasználói tér és különböző rendszermag-összetevők közötti felület. A ***procfs*** hosszú történetének köszönhetően rengeteg hulladékot halmozott fel, és az új felhasználó-rendszermag alkalmazásprogramozói felületeket arra ösztönzi, hogy különböző fájlrendszerekként valósuljanak meg. Ezek a fájlrendszerek a rendszermagra épülnek, és nincs valódi kiterjesztett jellemző támogatás. Ilyen például az ***usbfs***, a ***sysfs*** és a ***selinuxfs***.
Az ilyen kiterjesztett jellemzőket nem támogató rendszerek különböző címkézési viselkedésmódokat használnak, az egyes fájlrendszer-fajták biztonsági rendjéhez igazodva.
Az átmeneti ***SIDs*** címkézési viselkedésmód a ***devpts***, ***tmpfs*** és ***shmfs*** fájlrendszereknél használatos. Az ezekben a fájlrendszerekben található fájlok igény szerint a rendszermagban kerülnek címkézésre, az adott feladat és az érvényben lévő fájlrendszer biztonsági környezete alapján.
A ***devpts*** egy különleges átmeneti ***SIDs*** fájlrendszer. Kiterjesztett jellemző alkalmazásprogramozói felület elérést biz-

tosít a *pty*-khez egy álkiterjesztett jellemző biztonsági kezelőn keresztül. A kitüntetett alkalmazások, mint amilyen az `sshd` ezt a szolgáltatást a *pty*-k újracímkézésére használja, az átmeneti *SID* címkék felülbírálásával.
A *SIDs* címkézési viselkedésmód feladat egyszerűen megcímkézi az aktuális feladattal azonos biztonsági környezetben lévő fájlt. A *pipefs* és a *sockfs* fájlrendszerben létrehozott csővezetékek illetve csatlakozópontok esetében használatos.
A `genf_contexts` címkézési viselkedési módot olyan fájlrendszereknél használják, amelyek alkalmatlanok az *xattr*, átmeneti *SIDs* és feladat *SIDs* címkézésre. A biztonsági rendben a biztonsági környezet címkék fájlrendszer/elérési út párokhoz vannak rendelve. Az elérési út összetevő célja, hogy lehetővé tegye a fájlrendszer finomabb léptékű címkézését. Ez a szolgáltatás különösen a *procfs*-nél fontos, amely olvasható és írható rendszermag-adatok összevisszasága, beleértve a *systcl* felületet.
A legtöbb nem kiterjesztett jellemzős fájlrendszer a `genf_contexts` címkézést használja, általában úgy, hogy az egész fájlrendszer egyetlen biztonsági környezetre van állítva. Ennek gyakori példája a *sysfs*, a *vfat*, az *nfs* és az *usbdevfs*.

Befűzési pont címkézése

A 2.6.3-mas rendszermagba újonnan került szolgáltatás a befűzési pont címkézése, vagy másként a környezetbefűzés. Ennek az a fő célja, hogy lehetővé tegye, hogy az egész fájlrendszer biztonsági környezetét meg lehessen határozni egy befűzési beállítással. A befűzési pont címkézése bármilyen fájlrendszerre megvalósítható, és felülbírálja a rendes címkézési viselkedésmódot.
A befűzési pont címkézésének egy meghatározott felhasználási módja annak lehetővé tétele, hogy a különböző *NFS* befűzések külön-külön lehessen címkézni, befűzési időben. Olyan fájlrendszerek általános esetenkénti címkézéséhez is hasznos, amelyek nem támogatják a kiterjesztett jellemző biztonsági címkézést, valamint máshol címkézett kiterjesztett jellemző címkézésű fájlrendszerek címkézésére. Ez utóbbi például a törvényszéki munkában lehet hasznos.
A címke nélküli öröklött fájlrendszereket is lehet, hogy *SELinuxra* alkalmas operációs rendszereken kell befűzni. Ugyan a fájlrendszertípus támogatja a kiterjesztett jellemző biztonsági címkézést, esetleg nem akarunk maradandó biztonsági környezet címkéket adni a fájlrendszerekhez. A befűzési pont címkézése segítségével rendszermag alapú címkéket rendelhetünk hozzá, amelyek nem íródnak a lemezre.
Mivel a befűzési pont címkézése új szolgáltatás, és nem elég alaposan dokumentált, vegyük egy kicsit részletesebben. Amikor a *SELinux* engedélyezett a rendszermagban, három új lehetőség válik elérhetővé a befűzési pont címkézése számára:

- `context`: A fájlrendszer összes fájlját és magát a fájlrendszert a meghatározott biztonsági környezet szerint címkézi. A */proc/self/attr/fscreate* alkalmazásprogramozói felületet, amelyet fent taglaltunk, a fájlrendszer figyelmen kívül hagyja. Ez felülbírálja a meglévő címkézési viselkedésmódot, és a befűzési pont címkézésére változtatja. Ezzel a lehetőséggel a fájlrendszer címkéi a felhasználó számára kizárólag olvashatóak, annak ellenére, hogy a házirend által meghatározott átmenetek még működnek azokon a fájlrendszereken, amelyek támogatják a kiterjesztett jellemző biztonsági címkézést.
- `fscontext`: A teljes fájlrendszer címkéjét (azaz a fájlrendszert magát) beállítja a meghatározott biztonsági környezetre állítja. Ez lehetővé teszi a fájlrendszerek finomabb léptékű vezérlését, azáltal, hogy hozzájárul, hogy a címkéket a befűzés alapján lehessen beállítani, a házirendben meghatározott fájlrendszer alapú beállítás helyett. Mivel a `context` lehetőség is ezt a működést valósítja meg, nem lehet együtt használni a két lehetőséget. Ez a beállítás csak olyan fájlrendszereken működik, amelyek támogatják a kiterjesztett jellemző biztonsági címkézést. A teljes fájlrendszer biztonsági környezetei egy adott fájlrendszeren belül, a fájlok létrehozása során hozott hozzáférés-szabályozási döntéseknél, fájlrendszerek befűzésekor és leválasztásakor, fájlrendszer-jellemzőkhöz való hozzáféréskor és maga a fájlrendszer újracímkézésekor használatosak.
- `defcontext`: Beállítja a címkézetlen fájlok alapértelmezett biztonsági környezetét a házirendben meghatározott érték helyett. Ahogy az `fscontext` lehetőségnél is,

Biztonsági mentés és visszaállítás

A *SELinuxot* használó rendszergazdák számára számos változó feladat egyike a biztonsági mentés és visszaállítás. Amikor archívumot hozunk létre, hogyan őrződnek meg a biztonsági környezet címkék az archívumon belül? A válasz az igen rugalmas `star(1)` segédprogram használata, amely kiterjesztett jellemző támogatással rendelkezik.
Az archívumok biztonsági környezet címkékkel való befolyásolásához használjuk az `xattr` parancsot. Archívumok létrehozásakor meg kell határozni az `exustar` formátumot. Például:

```
$ star -xattr -H=exustar -c -f cups-log.star
➥/var/log/cups
```

létrehozza a */var/log/cups* könyvtár archívumát, a fájlokban megőrizve a biztonsági környezet címkéket.
A kicsomagoláshoz egyszerűen használjuk a `xattr` kapcsolót:

```
$ star -xattr -x -f cups-log.star
$ ls -Z var/log/cups/
-rw-r--r--+ root     sys
➥system_u:object_r:cupsd_log_t error_log
-rw-r--r--+ root     sys
➥system_u:object_r:cupsd_log_t error_log.1
```

Mint látható, a biztonsági környezet címkék megmaradtak.

csak olyan fájlrendszerekkel működik, amelyek támogatják a kiterjesztett jellemző címkézést, és érvénytelen, ha a `context` lehetőség van meghatározva, mivel az is ezt a működést valósítja meg.

A rendszermagban a ***SELinux*** a `mount(2)` alatt elemzi és kiiktatja a biztonsági befűzési beállításokat, rendes beállításokat átadva a fájlrendszerfüggő kódon keresztül. A hagyományos fájlrendszereknél nem kell ügyelni a biztonsági beállításokra, így nincs szükség a módosításokra. Ez azért lehetséges, mert a legtöbb fájlrendszer általában név/érték párokat használ befűzési lehetőségként, amelyet a ***SELinux*** könnyen befolyásolhat.
A bináris befűzési beállítás adatokkal rendelkező fájlrendszerek esetében, amilyen a ***NFS***, az ***SMBFS***, az ***AFS*** és a ***Coda***, különleges módon kell eljárni. Ezek közül csak az ***NFSv3*** támogatott a ***SELinux*** fejlesztésének mostani szakaszában.
Íme egy példa, a `context` lehetőség működésére, mivel valószínű, hogy a három befűzési lehetőség közül ez a leggyakrabban használt. Egy naplófájlokat tartalmazó hajlékony lemez érkezett az asztalunkra, és szeretnénk befűzni a ***SELinux*** gépre, néhány elemző programot futtatni rajta. Amiatt, ahogy a házirend meg van határozva, a fájlokat a `system_u:object_r:var_log_t` címkével kell ellátni, hogy a naplóelemző program megfelelően működjön. Ha ezzel a módszerrel fűzünk be, elkülöníthetjük az adatot a lemezen, lehetővé téve a ***SELinuxnak***, hogy megvédje egymástól az operációs rendszert és a lemez tartalmát.
Fűzzük be a lemezt:

```
$ mount -v -t vfat
➥-o context=system_u:object_r:var_log_t
➥/dev/fd0 /mnt/floppy
/dev/fd0 on /mnt/floppy type vfat
(context=system_u:object_r:var_log_t)
Mit mond az ellenőrzési napló?
SELinux: initialized (dev fd0, type vfat), uses
mountpoint labeling
```

Ez az üzenet ígéretesnek tűnik. Következőnek ellenőrizzük, hogy a lemez fájljai úgy vannak címkézve, ahogy gondoltuk. Rendszerint a `getfilecon(1)` hívást alkalmaznánk, de a `getfattr(1)` egyértelműbb hibaüzeneteket ad:

```
$ getfattr -n security.selinux
/mnt/floppy/access_log
/mnt/floppy/access_log: security.selinux:
➥Operation not supported
```

Mi történik? Egy `ls -Z` parancs is megmutatja, hogy a fájl üres biztonsági környezettel rendelkezik:

```
$ ls -Z /mnt/floppy/access_log
-rwxr-xr-x+  root  root  (null)
➥/mnt/floppy/access_log
```

A hajlékonylemezen található ***vfat*** fájlrendszer nem rendelkezik kiterjesztett jellemző támogatással, és a biztonsági környezet címkézése tisztán a rendszermagon belül történik. Kiderül, hogy a rendszermagon belüli címkézés jól működik, de a felhasználói tér eszközei nem képesek megjeleníteni a kiterjesztett jellemző alkalmazásprogramozói felületen található címkéket. Ez az aktuális kiterjesztett jellemző megvalósítás korlátja, amelyet még elegánsan meg kell oldani.
Van viszont egy trükkös módja, hogy megnézzük, mik a fájlok címkéi – az ellenőrzési napló alkalmazásával, amely hozzáférési üzenetek naplózásakor mindig rögzíti a célobjektum biztonsági környezetét.
A `getfattr(1)` használata a következő ellenőrzési bejegyzést idézte elő:

```
avc:  denied  { getattr } for  pid=12354
exe=/usr/bin/getfattr
path=/mnt/floppy/access_log dev=fd0 ino=132
scontext=root:staff_r:staff_t
tcontext=system_u:object_r:var_log_t tclass=file
```

Tehát a fájl címkézése helyes (`system_u:object_r:var_log_t`), amely a `context` befűzési lehetőségen keresztül került a befűzés parancshoz.

Jövőbeni munka

Most is lehet ugyan biztonsági környezeti címkéket rendelni az ***NFS***-sel befűzött fájlrendszerekhez, de azok csak helyben működnek, a rendszermagon belüli hozzáférés-szabályozási döntéseknél. A címkék nem kerülnek átvitelre a hálózaton a fájlokkal együtt. Ezen a területen fejlődés tapasztalható az ***NFSv2/v3*** protokollokon és kódon történt ***SELinuxra*** szabott módosításoknak köszönhetően. Hosszabb távon az ***NFSv4***-be várhatóan bekerül a hálózati címkézés nevesített jellemzőkön keresztül, amelyek részei a kiterjedtebb ***NFSv4*** leírásnak. Ez lehetővé tenni, hogy az ***NFS*** ügyfél is és a kiszolgáló is megvalósítsa a ***SELinux*** biztonságot a hálózati fájlokon. Más hálózati fájlrendszerek számára szintén hasznos lenne a támogatás, ahogy a ***Trusted BSD SELinux*** kapujával való együttműködés is.

Linux Journal 2004. október, 126. szám

James Morris (jmorris@redhat.com) egy Sydney-i ausztrál rendszermag fejlesztő, aki jelenleg a Red Hat-nek dolgozik, Bostonban. Rendszermag-karbantartó a SELinuxban, a hálózati és titkosító API-ban, valamint LSM fejlesztő és az Emeritus Netfilter Core csapat tagja.

Az SQL és az Nmap szövetsége: hatékonyság és kényelem

Ha az Nmap-et használjuk a hálózaton lévő gépek biztonságának ellenőrzésére, akkor érdemes a MySQL-re bíznunk a trendek és változások nyomon követését.

Levelet váltottam nemrég valakivel, akinek rendszeres kapupásztázást kell végeznie a hálózatán a sebezhető pontok felderítése végett. Az általa használt kapupásztázó segédprogram az ***Nmap***, ami azonban maga nem teszi könnyen kezelhetővé az általa szolgáltatott adatokat. Ez bizony már egy teljesen más terület. Néhány héttel később elkészült az ***Nmaphez*** egy olyan folt, amely lehetővé tette az eredmények ***MySQL*** adatbázisban történő rögzítését. Bár az ***Nmap*** támogatja a mind a géppel elemezhető, mind pedig az ***XML*** kimeneti formátumot, az ***SQL***-adatbázisba történő naplózás kényelme és hatékonysága messze meghaladja az ***XML***, de még a géppel elemezhető kimeneti formátum képességeit is. Hogy csak egy dolgot említsek, az ***nmapsql*** nem tartalmaz külön lépést a parancshéjban a kimenet utófeldolgozó egységbe való áttöltésére.
Az ***nmapsql*** egy közvetlen folt a ***Fyodor*** által megalkotott tiszteletre méltó ***Nmap*** hálózatpásztázó program 3.48-as változatához. (Amikor ezt írom, már megjelent az ***Nmap 3.50***-es változata, amelyhez egy frissített ***nmapsql*** is letölthető a program honlapjáról.) A folt tartalmaz ***MySQL***-támogatást, de az eredmények egyszerű átvitelén kívül sokkal többre is képes: célmegjelölést, pásztázó-kijelölést, és egyszerű összehasonlító kiértékelést (trending) is lehetővé tesz. Az adatok ***SQL***-adatbázisba való áttöltésével egy sereg új feladat elvégezhető. Az ***nmapsql*** a ➲ ***sourceforge.net/projects/nmapsql*** címről tölthető le. A program az adatkezeléshez jelenleg a ***MySQL*** ügyfélprogramjának felhasználói felületét alkalmazza.
Mivel a hálózati biztonságért felelős rendszergazdák nem feltétlenül adatzsonglőrök, az ***nmapsql*** tervezésekor fontos szempont volt az egyszerű kezelhetőség. A folt működése elég egyszerű ahhoz, hogy a hálózatvizsgálat eredményének legfontosabb információi egyetlen táblából kinyerhetőek legyenek. Az egyszerűség az oka annak is, hogy az IP-címek tárolása formázatlan szövegként és nem `inet_aton()` jelölésben történik. Ismerem a szövegkezelés teljesítményben megmutatkozó hátrányait, de most a kis adathalmazok kényelmes kezelésének bemutatására összpontosítunk. Nagy adathalmazok esetén, amikor a sebesség kérdése kritikus szempont, a célmegjelölő címkék, futási és pásztázó-azonosítók állnak rendelkezésre a numerikus keresés számára.

A cikkben először egy ***SQL***-t használó hálózatpásztázással foglalkozunk, amellyel egy hálózat nyitott kapuit és élő célgépeit derítjük fel. Ezután az ***SQL***-ben összegyűjtött adatokra vetünk egy pillantást és megvizsgáljuk, milyen módszerekkel lehet a kapott eredményeket összehasonlítani.

A beállítási lehetőségek

Az ***nmapsql*** tevékenységét az aktuális felhasználó saját könyvtárában lévő ***~/nmapsql.rc*** fájl beolvasásával kezdi. Ez azt jelenti, hogy ha az `nmapsql` futtatása előtt a `su -` parancsot használjuk a rendszergazdai jogosultság megszerzéséhez, a beolvasott fájl a ***/root/nmapsql.rc*** lesz. Egyelőre mindössze négy tétel szerepel az ***nmapsql.rc*** fájlban, mindegyik külön sorban, `adat=érték` formátumban, ahogy sok más programnál is láthatjuk. A tételek a következők:

```
server=localhost,
db=nmaplog,
user=nmap,
passwd=scanamanga
```

A `server` annak a gépnek a ***DNS***-neve, amelyen a ***MySQL*** fut, a `db` pedig ezen a kiszolgálón található adatbázis neve. A `user` és `password` tételek az adatbázishoz való csatlakozáshoz szükséges felhasználói név és jelszó. Az itt szereplő felhasználónak az adatbázison legalább `SELECT`, `INSERT` és `UPDATE` jogosultságokkal kell rendelkeznie.
Az ***Nmap*** által biztosított parancssori lehetőségeket az ***nmapsql*** négy további paraméterrel egészíti ki:

```
--mysql
--runid
--targetid
--scannerid
```

Ha az ***nmapsql*** futtatható állományát ezek nélkül a paraméterek nélkül indítjuk, a működése teljesen megegyezik az ***Nmap*** működésével. Ezen beállítások egyike sincs hatással az ***Nmap*** létező paramétereire, így semmi akadálya annak,

1. lista Az Nmap kimenetének egy részlete

```
Starting nmap 3.48 (
http://www.insecure.org/nmap/ )
at 2003-12-14 10:00 SGT
Insufficient responses for TCP sequencing (1),
OS detection may be less accurate
Interesting ports on 192.168.10.0:
(The 1656 ports scanned but not shown below are
in state: closed)
PORT    STATE SERVICE VERSION
80/tcp open  http?
Device type: print server
Running: Linksys embedded
OS details: Linksys EtherFast print server

Interesting ports on wap.hasnains.com
(192.168.10.1):
(The 1654 ports scanned but not shown below are
in state: closed)
PORT    STATE SERVICE VERSION
21/tcp open  ftp?
23/tcp open  telnet?
80/tcp open  http
Device type: broadband router
Running: Zyxel ZyNOS
OS details: ZyXEL Prestige 700/Netgear MA314
broadband router
```

hogy ugyanannak a hálózatvizsgálatnak az eredményét egyidejűleg SQL-adatbázisba és gépileg elemezhető kimenetre irányítsuk.
A `--mysql` kapcsolót a többi ***nmapsql*** kapcsoló nélkül használva a parancssorban olyan ***MySQL***-naplózás indul el, amelynek során minden címke- és azonosító-kijelölés önműködően történik. Az összes többi ***nmapsql*** paraméter feltételezi a `--mysql` használatát. Az önműködő kijelölés során a folyamat veszi a megfelelő tábla legnagyobb értékét és egyesével halad az értékeken.
A pásztázó azonosító megadásának lehetőségét bevezető `--scanner-id xxx` paraméter, amelyben az `xxx` az azonosító értékét jelöli, olyan helyzetek esetében alkalmazhatók, amikor több pásztázó kerül üzembe helyezésre, például egy több alhálózattal rendelkező környezetben. A pásztázó-azonosító (`scanner ID`) és a futásidejű azonosító (`runtime ID`) a `portstat` táblában tárolódik lehetővé téve a pásztázást végző gép számára az eredményhalmazok különválasztását. Az alábbi lekérdezés használatával így egyszerűen különválaszthatnánk például a tízes pásztázó eredményeit:

```
mysql> select * from portstat
    -> where scannerid = 10 and runid = 100;
```

A `--run-id xxx` paraméter az ***nmapsql*** adott futtatásához rendelt azonosító megadására szolgál. Amennyiben nem használjuk ezt a lehetőséget, a rendszer önműködően hozzárendelt azonosítót használ. Ha a megadott futtatás-azonosító már szerepel az adatbázisban, ismételten felhasználásra kerül. Ez a lehetőség biztosítja, hogy különböző futtatások eredményeit kényelmesen csoportosíthassuk egy futtatási azonosító alatt.
A futtatás-azonosító (`runtime ID`) és a hozzá kapcsolódó információk tárolására a `runlist` tábla biztosít helyet. A használt táblák ismertetését „Az nmaplog által használt táblák” című kiemelt részben láthatjuk. A vizsgálat befejeztével frissül a futásidejű információk egy része, köztük a parancssorban megadott összes lehetséges célpontok és a működő állapotban találtak száma. Hasonlóan tárolódik a pásztázó azonosítója (`scanner ID`) és az ehhez kapcsolódó információk a scanners táblában.
Minden vizsgált célpont egy címkét kap, az információ pedig a `targets` táblába kerül. A futási listához hasonlóan a `targets` tábla is két szakaszban kerül feltöltésre. Az első szakasz az IP-címeket és – amennyiben feloldható – a gépneveket gyűjti be, a második szakaszban pedig az `os_guessed` oszlop feltöltése történik. Pillanatnyilag nem tárolódik egy fel nem ismert operációs rendszer ujjlenyomat-információja, de a jövőben változhat a helyzet.
A `targets` táblában sosem jönnek létre kettőződések. Tapasztalatom szerint csak abban az esetben lehetnek kettőzött IP-alhálózatok, amennyiben egyik ügyféltől a másikhoz helyezzük át a rendszert. Ebben az esetben minden ügyfél esetén különböző adatbázisra van szükség.
A célazonosítókat (`target ID`) pillanatnyilag még nem használja a rendszer, de a felhasználónak a parancssorban lehetősége van saját célazonosító meghatározására. Amennyiben a megadott azonosító már létezik, az egyediség megőrzése érdekében a rendszer figyelmen kívül hagyja és egy rendszer által előállított azonosítót használ helyette.
Ha a parancssorban a `--target-id` után megadott célazonosító még nem szerepel a `targets` táblában, hozzárendelődik az adott cél IP-címéhez. Ha több rendszer számára történik a célmegadás, az első cél kapja meg a megadott célazonosítót, a rá következőkhöz pedig az eggyel növekvő azonosítók fognak tartozni.

Az alapok

Az ***nmapsql*** rögzíti a naplóban a futtatás dátumát és időpontját, az ***Nmap***-et futtató felhasználó nevét, a futtató gép nevét, és a futtatáshoz rendelt azonosító számot. A két utóbbi adat lehetővé teszi, hogy az ***nmapsql***-t nagyobb rendszerekben is alkalmazzuk és alapot képeznek az egyes vizsgálatok összehasonlítására. A futtatás-azonosító (`runid`, ***runtime ID***) az adathalmazon belül mindig egyedi. Ha a megadott cél azonos, két vizsgálati eredmény között a futtatás-azonosító alapján tehetünk különbséget, lehetőség van azonban arra is, hogy több vizsgálati eredményt csoportosítsunk oly módon, hogy a pásztázáshoz a `--run-id` kapcsoló segítségével egyetlen futtatás-azonosítót adunk meg. Vizsgáljuk meg például az `nmapsql` alábbi indítását:

```
$ nmap -A --mysql --runid 100 192.168.10.1/24
```

A parancs a naplózási lehetőséget engedélyező `--mysql` kapcsolóval indítja el az ***Nmap***-et, a futtatás-azonosító értékének

2. lista Egy adott pásztázás eredménye

target_ip	d	t	port	protocol	state	runid
192.168.10.0	2003-12-14	10:00:37	80	tcp	open	100
192.168.10.1	2003-12-14	10:00:37	21	tcp	open	100
192.168.10.1	2003-12-14	10:00:37	23	tcp	open	100
192.168.10.1	2003-12-14	10:00:37	80	tcp	open	100
192.168.10.44	2003-12-14	10:00:37	22	tcp	open	100
192.168.10.44	2003-12-14	10:00:37	111	tcp	open	100
192.168.10.44	2003-12-14	10:00:37	3306	tcp	open	100
192.168.10.44	2003-12-14	10:00:37	6000	tcp	open	100
192.168.10.64	2003-12-14	10:00:37	135	tcp	open	100
192.168.10.64	2003-12-14	10:00:37	139	tcp	open	100
192.168.10.64	2003-12-14	10:00:37	445	tcp	open	100
192.168.10.64	2003-12-14	10:00:37	1024	tcp	open	100
192.168.10.64	2003-12-14	10:00:37	1025	tcp	open	100
192.168.10.64	2003-12-14	10:00:37	1031	tcp	open	100
192.168.10.64	2003-12-14	10:00:37	5000	tcp	open	100
192.168.10.64	2003-12-14	10:00:37	5101	tcp	open	100
192.168.10.64	2003-12-14	10:00:37	6000	tcp	open	100
192.168.10.65	2003-12-14	10:00:37	135	tcp	open	100
192.168.10.65	2003-12-14	10:00:37	139	tcp	open	100
192.168.10.65	2003-12-14	10:00:37	445	tcp	open	100
192.168.10.65	2003-12-14	10:00:37	1025	tcp	open	100
192.168.10.65	2003-12-14	10:00:37	5000	tcp	open	100

100-at ad meg és megvizsgálja a 192.168.10.1/24 hálózatot. Amennyiben ez az `nmapsql` első futtatása, egy összehasonlítási alap jön létre a hálózat számára, amelyhez a későbbi futtatások eredményeit hasonlíthatjuk. Az ***nmapsql*** emellett önműködően létrehoz a futtató gép számára egy bejegyzést, ebben az esetben a `192.168.10.44` értékkel, amelyet hozzárendel a `scanners` tábla egy `scanner_id` mezőjéhez. Az *1. listán* a futtatás konzolkimenetének részlete látható.

A cél meghatározása ebben az esetben a teljes C-osztályú alhálózatot jelenti. Az ***nmapsql*** önműködően egyedi azonosítókat rendel a hálózaton talált minden működő géphez, és további információkat is tárol a `hoststats` táblában. Önmagában már ez a tábla is kiinduló eszköz lehet egy kapupásztázás-összehasonlításhoz.

Vizsgáljuk meg röviden, hogy milyen adatok kerültek rögzítésre. Ehhez először is bejelentkezünk a ***MySQL*** ügyfélprogramjába és csatlakozunk az ***nmapsql.rc*** fájlban megadott adatbázishoz. Ezután a következő lekérdezést futtatjuk:

```
$ mysql nmaplog -p
mysql> select target_ip, d, t, port, protocol,
    -> state, runid from portstat
    -> order by target_ip, d, t ;
```

A lekérdezés a *2. listán* látható táblázatot adja eredményül, egy tetszetős listát, amelyben az adatok a cél IP-címe, a dátum és idő szerint rendezett sorrendben jelennek meg. Látható, hogy a `runid` oszlop tartalma minden esetben 100, ahogy azt a parancssorban megadtuk.

A következő lekérdezés futtatásával a *3. listán* látható, a `192.168.10.44` cím nyitott kapuira vonatkozó információt kapjuk:

```
mysql> select target_ip, d, t, port, protocol,
    -> state, runid from portstat
    -> where target_ip = ´192.168.10.44´
    -> order by d, t;
```

Ha a kapott négy sort hozzáillesztjük az 1. lista `192.168.10.44` címre vonatkozó szakaszához, azonnal felismerhetjük a köztük fennálló kapcsolatot. Látszik, hogy a `portstat` tábla önmagában is tartalmazza az ***Nmap*** által kapupásztázásra vonatkozóan szolgáltatott összes információt. Természetesen ha már több futtatási eredménnyel rendelkeznénk, a fenti lekérdezés a `192.168.10.44` címre vonatkozó minden adatot kilistázná.

Tegyük fel, hogy két nap, egy hét, esetleg egy hónap elteltével újra megvizsgáljuk a hálózatot és az eredményeket vizuálisan is össze szeretnénk hasonlítani. A `runlist` táblára vetett első pillantásra látható, hogy a két nappal későbbi vizsgálatnak a 102-es futtatás-azonosító felel meg. Ennek az információnak a felhasználásával már írhatjuk is a lekérdezésünket:

```
mysql> select target_ip, d,t,port, protocol,
    -> state from portstat
    -> where target_ip = ´192.168.10.44´
    -> and runid = 102 order by d,t,port;
```

3. lista Egy adott gép vizsgálatának eredményei

```
+-----------------+-------------+-----------+---------+-----------+---------+----------+
| target_ip       | d           | t         | port    | protocol  | state   | runid    |
+-----------------+-------------+-----------+---------+-----------+---------+----------+
| 192.168.10.44   | 2003-12-14  | 10:00:37  |     22  | tcp       | open    | 100      |
| 192.168.10.44   | 2003-12-14  | 10:00:37  |    111  | tcp       | open    | 100      |
| 192.168.10.44   | 2003-12-14  | 10:00:37  |   3306  | tcp       | open    | 100      |
| 192.168.10.44   | 2003-12-14  | 10:00:37  |   6000  | tcp       | open    | 100      |
+-----------------+-------------+-----------+---------+-----------+---------+----------+
```

4. lista A 192.168.10.44 cím eredményei egy két nappal később folytatott vizsgálat után

```
+--------------+------------+----------+---------+-----------+--------+
| target_ip    | d          | t        | port    | protocol  | state  |
+--------------+------------+----------+---------+-----------+--------+
| 192.168.10.44 | 2003-12-16 | 00:47:16 |     22  | tcp       | open   |
| 192.168.10.44 | 2003-12-16 | 00:47:16 |    111  | tcp       | open   |
| 192.168.10.44 | 2003-12-16 | 00:47:16 |   3306  | tcp       | open   |
| 192.168.10.44 | 2003-12-16 | 00:47:16 |   6000  | tcp       | open   |
+--------------+------------+----------+---------+-----------+--------+
```

Könnyen összehasonlíthatjuk a *4.* és *5. lista* eredményeit és kiszűrhetjük a különbségeket. Természetesen a két adathalmaz összehasonlítását egy program is elvégezheti, amely összefoglalja számunkra a különbségeket.
A korábban említett `hoststat` tábla minden működő gép információit tartalmazza. Az `open_ports` oszlop egyszerű összegzést nyújt a nyitott kapuk számáról. Ahhoz, hogy a célgépünk nyitott kapujának egy soros információit megkapjuk, a következő lekérdezést kell futtatnunk:

```
mysql> select ip,d,t,open_ports, ports_scanned,
    -> runid from hoststats where order by ip, d, t;
```

(Az `order by` záradékot a továbbfejlesztés érdekében írtuk a lekérdezéshez.) Az `open_ports` oszlopot a `date/time` és runid oszlopokkal együtt vizsgálva vázlatos képet kapunk a nyitott kapuk állapotának egy biztonyos időn belüli változásairól. A targets tábla az összes előforduló célgéppel kapcsolatos információt tartalmazza, IP-címenként egy-egy sort. Ez az egyetlen tábla, amelyben a gép neve (amennyiben meghatározható) és az ***Nmap*** által feltételezett operációs rendszer tárolásra kerül. Nézzük, milyen információkat nyerhetünk a vizsgált címről:

```
mysql> select  * from targets
    -> where ip = ´192.168.10.44´;
```

Láthatjuk, hogy az `OS_guessed` mező tartalma jelen esetben `Linux Kernel 2.4.0-2.5.20`, a `hostname` (gépnév) oszlop pedig az `ophelia.hasnains.com` szöveget tartalmazza (szeretem Shakespeare tragikus hősnőit). Most, hogy már lényegében minden részlet a birtokunkban van, írjunk egy olyan lekérdezést, amely összegyűjtve tartalmazza a vizsgálat tárgyát képező gépre vonatkozó összes adatot.

```
mysql> select r.runid, r.d, r.t, t.ip, t.host,
    -> t.os_guessed, p.port, p.protocol,
       ➥p.service,
    -> p.state, p.fullversion from runlist r,
    -> targets t, portstat p
    -> where r.runid = 100 and p.target_ip = t.ip
    -> and p.runid = r.runid
    -> order by r.runid, r.d, r.t, t.ip;
```

Helyszűke miatt nem mutatjuk meg a lekérdezés kimenetét, próbáljuk ki nyugodtan saját magunk. Használhatnánk jelentéskészítő programot is az eredmény célpontok szerinti csoportosításához. Ha tetszetősebb kimenetet szeretnénk, hatékonyabb fegyvereket kell bevetnünk, például a ***PHP*** vagy a ***Perl*** tehet megfelelő erre a célra.
A jelentések közül az egyik leghasznosabb az egyes célpontok nyitott kapuinak változását szemléltető kimutatás. Vegyük például azt az esetet, amikor a vizsgált gépen bezáródott a 111/TCP kapu, de a 23/TCP nyitott állapotba került. Ebben az esetben a hoststats tábla open_ports oszlopa még mindig négy kaput fog mutatni, pedig a részletekben változás következett be. A megfelelő program könnyedén kiválogathatja a különbsége(ke)t egy jelentés számára.

Hasznos kimutatások

Az alábbi, leggyakrabban használt lekérdezés arra a kérdésre ad választ, hogy egy bizonyos célpont esetén mely kapuk vannak nyitott állapotban:

```
mysql> select d, t, port, protocol, state,
    -> fullversion from portstat
    -> where target_ip =  ´192.168.10.44´
    -> order by d,t,port;
```

5. lista Egy adott gépre vonatkozó összesített eredmények

```
+------------+----------+--------------+---------------------+---------------------------+------+----------+---------+-------+-------------------------------+
| d          | t        | ip           | host                | os_guessed                | port | protocol | service | state | fullversion                   |
+------------+----------+--------------+---------------------+---------------------------+------+----------+---------+-------+-------------------------------+
| 2003-12-14 | 10:00:37 | 192.168.10.44 | ophelia.hasnains.com | LinuxKernel 2.4.0 - 2.5.20 |   22 | tcp      | ssh     | open  | OpenSSH 3.5p1 (protocol 1. 99) |
| 2003-12-14 | 10:00:37 | 192.168.10.44 | ophelia.hasnains.com | LinuxKernel 2.4.0 - 2.5.20 |  111 | tcp      | rpcbind | open  | 2 (rpc #100000)               |
| 2003-12-14 | 10:00:37 | 192.168.10.44 | ophelia.hasnains.com | LinuxKernel 2.4.0 - 2.5.20 | 3306 | tcp      | mysql   | open  | MySQL4.0.16-standard-log      |
| 2003-12-14 | 10:00:37 | 192.168.10.44 | ophelia.hasnains.com | LinuxKernel 2.4.0 - 2.5.20 | 6000 | tcp      | X11     | open  | (access denied)               |
+------------+----------+--------------+---------------------+---------------------------+------+----------+---------+-------+-------------------------------+
```

Egy másik gyakran használt kimutatás azt mutatja meg, hogy egy célgép bizonyos kapuja nyitva volt-e valamikor egy korábbi időpontban: „Nyitva volt az SSH a `192.168.10.44` címen két héttel ezelőtt?” Feltéve, hogy az ***nmapsql*** telepítve volt már és a ***crontab*** rendszeres időközönként le is futtatta, a válasz a következő lekérdezéssel kideríthető:

```
mysql> select d, t,target_ip, port,protocol,
    -> service, state, fullversion from portstat
    -> where port = 22 and protocol = “tcp”
    -> and state = “open”
    -> d = date_sub( curdate(), interval 14 day)
    -> order by d, runid, target_ip ;
```

Természetesen egy adott napon több `nmapsql` futtatásunk is lehet, emiatt szerepel az order by záradék. Ha az eredményhalmaz előállításához olyan más gyártótól származó eszközt használunk, mint a ***PHP*** vagy a ***Perl***, a runlist táblából kell a pontos időkerethez tartozó futásazonosítót megszereznünk és ezt felhasználva kell a kiválasztott célra vonatkozó információkat lekérdeznünk.
Egy másik hasznos lekérdezéssel egy adott hálózat azon célpontjait szűrhetjük ki, amelyek bizonyos kapuja nyitott állapotban van: „A `192.168.10/24` alhálózat hány gépén van nyitva a 80/TCP kapu?” A keresett eredményt a következő lekérdezéssel kaphatjuk meg:

```
mysql> select runid, d, t, target_ip, port,
    -> protocol, state from portstats
    -> where port = 80 and protocol = ´tcp´
    -> and state = ´open´
    -> and target_ip like ´192.168.10.%´;
```

A szöveges megfeleltetés nem igazán alkalmas az alhálózat azonosítására, de a példa jól szemlélteti a megoldás lényegét.

Különböző adatbázisok használata

Számos esetben – például amikor egy szakértő több hálózaton dolgozik felváltva – kívánatos, hogy meg tudjuk változtatni az adatbázis nevét (ami esetleg lehet az ügyfelünk neve), így a különböző helyek adatai nem keveredhetnek össze egymással. A cikk írásának idején ezt úgy tehetjük meg, hogy módosítjuk az nmaplog által az adatbázis-tulajdonságok megadására szolgáló ***~/nmapsql.rc*** fájl ***db=nmaplog*** bejegyzést.
Az adatbázis menet közben való megváltoztatásához az ***~/nmapsql.rc*** fájlban lévő nmaplog helyére írjuk be a megfelelő nevet, és győződjünk meg arról, hogy a fájlban megadott felhasználó rendelkezik a megfelelő jogosultságokkal az így beállított adatbázishoz. Ezután töltsük be az adatbázis-sémát az új adatbázisba. Ha az új adatbázis neve newnmap, akkor az alábbi egysoros paranccsal tölthetjük át a sémát:

```
$ mysql newnmap < nmaplog.sql
```

Mégsem javaslom azonban a különböző adatbázisok használatát, sokkal egyszerűbb az adatokat egy fájlba kimásolni és ezután az nmaplog adatbázisba egy üres sémát betölteni. A következő parancssorokkal hajthatjuk végre ezt a műveletet:

```
$ mysqldump nmaplog > newnmap.sql
$ mysql nmaplog < nmaplog.sql
```

Az adatbázis-jogosultságok beállításától függően megeshet, hogy a parancsok fenti futtatásához létre kell hoznunk egy ***MySQL***-felhasználót és jelszót.

Alkalmazási módok

Ha csak ritkán és néhány gépen alkalmazzuk az ***nmapsql***-t, nehezebben ismerhetőek fel a program hasznos tulajdonságai. A program nagyobb környezetekben, több alhálózat és több tucatnyi vizsgálati célpont esetén mutatja meg az igazi tudását. A legegyszerűbb alkalmazási eset természetesen az, amikor az ***nmapsql*** és a ***MySQL*** kiszolgáló ugyanazon a gépen helyezkedik el, ami lehet például egy hálózati szakértő hordozható gépe, amellyel hálózatról hálózatra jár. Mivel a legtöbb hálózat tűzfalas védelem alatt áll és RFC 1918-as címzést használ, egy hálózatok közt kóborló hordozható gépen lévő targets táblában nagy a valószínűsége, hogy egyes IP-címek többször is előfordulnak. Ebben az esetben tehát át kell töltenünk az adatainkat és minden új környezetnél friss adatbázissal kell dolgoznunk.
Az ***nmapsql***-nek létezik más alkalmazási módja is: közepes méretű hálózatokban több pásztázó működhet a különböző alhálózatokban és a naplózásra használhatja ugyanazt a ***MySQL*** kiszolgálót, vagy nagy hálózatokban, ahol több egygépes rendszer (a ***MySQL*** és az ***nmapsql*** ugynazon a gépen fut) helyileg végzi a pásztázást és naplózást. Az ilyen rendszerekben kicsi az RFC 1918-as címek ketőződésének a valószínűsége. Igaz viszont, hogy a helyben történő pásztázás és naplózás valamint a központi kiszolgálón történő adatgyűjtés közti időkülönbség miatt nem rendelkezünk

Az nmaplog által használt táblák

Az *nmaplog* adatbázisban jelenleg nyolc tábla található, amelyek közül a fontosabbak az alábbiak:

- `TARGETS` – a célgépről tartalmaz információkat, többek közt a célazonosítót, IP-címet, a gép nevét és az *Nmap* által feltételezett operációs rendszert.
- `SCANNERS` – az *nmapsql*-t futtató gépről tartalmaz adatokat. A `portstat` táblához a `scan_id` mezővel kapcsolódik.
- `RUNLIST` – a felhasználó azonosítóját, az *Nmap* indításának dátumát és időpontját és a futtató gép információit tartalmazza. A felhasználó neve és azonosítója az /etc/passwd fájlból származik. A `scanner_id` mező a `scanners.scan_id` mezőhöz kapcsolódik.
- `PORTSTAT` – a kapupásztázás eredményeit tartalmazza. rögzítésre kerül az összes nmaplog jelzett kapu az állapotával (`open/close/filtered`, nyitott/zárt/szűrt) együtt.
- `HOSTSSTAT` – a célgép alapvető adatait, vagyis az nmapsql egyes futásainak időpontjaiban vizsgált összes, és az ebből nyitott állapotban lévő kapuk számát tartalmazza.

azonnali adatokkal. Mindkét helyzetre igaz, hogy a pásztázó-azonosító (scanner ID) jól használható az adatok szétválasztására.

Fejlesztési irányok

Gyakorló biztonsági szakértők – és el kell ismernem, néhány fekete kalapos számítógépes kalóz is – elismerik az nmapsql képességeit, amely nagyfokú elvárások kielégítésére is alkalmassá teszik. A projekt jelenlegi fejlesztési céljai közt szerepel, hogy a felhasználóknak lehetőségük legyen az *nmapsql* beállításait közvetlenül az *Nmap* felületén kersztül elvégezni, valamint egy olyan PHP-ben írt jelentéskészítő előtag létrehozása, amely megkíméli a végfelhasználót attól, hogy a lekérdezéseket a MySQL-be kelljen kézzel begépelnie. Ezek a kiegészítések pillanatnyilag fejlesztés alatt állnak. A távolabbi tervek közt szerepel a *Nessus* sebezhetőségvizsgálati eredményének ugyanabba az adatbázisba történő befoglalása, s ezzel egységes kezelőpult létrehozása a kapupásztázás és sebezhetőségi értékelés eredményei számára. A cél felé vezető lépésként az *nmapsql* honlapján találhatunk egy egyszerű formai elemzőt, amely alkalmas a *Nessus* ügyféllel létrehozott fájlok betöltésére.

Linux Journal 2004. október, 126. szám

Hasnain Atique (hatique@hasnains.com) a napos Szingapúrban él feleségével és három éves kislányával. Amikor lányával épp nem a Harry Pottert nézi, akkor próbál a hálózati gyűrűk ura lenni, ami néha sikerül is neki.

A Linux fájlrendszer biztonsága (2. rész)

A múlt hónapban megismerkedtünk a jogosultságok alapjaival. Most itt az ideje, hogy megvizsgáljunk néhány hasznos bitet a felhasználók kényelmes és biztonságos együttműködésének megteremtéséhez.

Múlt alkalommal az alapoktól indulva vizsgáltuk meg a fájl- és könyvtárjogosultságok rendszerét: megismertük a felhasználók és csoportok fogalmát valamint a fájlokra és könyvtárokra vonatkozó olvasási, írási és futtatási jogok beállításának és törlésének módját. Ebben a részben a jogosultságok néhány fejlettebb formájára irányítjuk figyelmünket, felderítjük a jogosultságok numerikus módszerét, az `umask` parancsot és a root jogosultságainak átadását a `su` és `sudo` parancsokkal. A mostani cikk a múlt havinál több középszintű információt tartalmaz, de remélhetőleg még akkor is érthető lesz, ha csak annyit tudunk a témáról, amit az előző alkalommal olvastunk.

A sticky bit

Idézzük fel a múlt hónapban tanulmányozott `extreme_casseroles` könyvtár listájának eredményét:

```
drwxr-x--- 8  biff drummers 288 Mar 25 01:38
➥extreme_casseroles
```

Biztosan emlékszünk még arra, hogy a könyvtárra vonatkozó csoport-jogosultságokat r-x értékre állítottuk be, vagyis a csoport által olvasható és futtatható értékre, úgyhogy a `drummers` csoport tagjai beléphetnek ebbe a könyvtárba és olvashatják a benne található recepteket. Tegyük fel, hogy dobos barátunk, `Biff`, szeretné, ha zenésztársai nem csak olvasni tudnák ezeket a recepteket, hanem a sajátjaikat is elhelyezhetnék itt. Ahogy az elmúlt alkalommal láthattuk, ehhez nem kell mást tennie, mint beállítani a könyvtár csoportra vonatkozó írási bitjét a következőképpen:

```
chmod g+w ./extreme_casseroles
```

Ezzel csak egy gond van: az írási jog magában foglalja az adott könyvtárban új fájl létrehozásának, illetve egy már létező fájl törlésének a jogosultságát is. Mi akadályozza meg az egyik dobos cimborát abban, hogy mások receptjeit letörölje? A ***sticky bit***, semmi más.
Régebben a ***sticky*** bitet arra használták, hogy a fájlt (programot) a memóriába olvassanak be, így a hívása esetén sokkal gyorsabban tudott betöltődni. A ***Linuxon*** azonban más a szerepe. Amikor egy könyvtár esetében beállítjuk a ***sticky*** bitet, akkor ezzel az adott könyvtárban korlátozzuk a felhasználók törlési lehetőségeit. Ez konkrétan azt jelenti, hogy a könyvtárban egy adott fájl letörléséhez tulajdonosi jogokkal kell rendelkeznünk vagy a fájl vagy a könyvtár esetében, még akkor is, ha a tulajdonos csoporthoz tartozunk, és a csoport írási lehetősége be van állítva.
A ***sticky*** bit beállításához az alábbi parancsot kell futtatnunk:

```
chmod +t könyvtárnév
```

A példánkban a parancs formája `chmod +t extreme_casseroles` lenne. Ha most az `ls` parancsot a `-d` kapcsolóval használva lekérdezzük magának a könyvtárnak a részletes jellemzőit, hogy a könyvtárhoz kötődő jogosultságok jelenjenek meg és nem a könyvtár tartalma, vagyis kiadjuk a `ls -ld extreme_casseroles` parancsot, az alábbi eredményt kapjuk:

```
drwxrwx--T 8  biff drummers  288  Mar 25 01:38
➥extreme_casseroles
```

Figyeljük meg a jogosultságokat jelző karakterlánc végi `T` betűt. Normál esetben itt egy `x` vagy `-` jelet várnánk attól függően, hogy a könyvtáron az egyéb felhasználók rendelkeznek-e írási joggal. A `T` azt jelöli, hogy a könyvtáron az egyéb felhasználóknak nincs futtatási joga és a ***sticky*** bit be van kapcsolva. A korlátozás hatásának bemutatásához tegyük fel, hogy az `extreme_casseroles` könyvtár tartalmának a kiíratása az *1. listán* látható eredményt adja.
Tegyük fel továbbá, hogy a `crash` nevű felhasználónak nem tetszik a `pineapple_mushroom_surprise.txt` fájl és ezért megpróbálja letörölni. Arra számít, hogy ez sikerülni is fog neki, hiszen a `drummers` csoporthoz tartozik, és a csoport rendelkezik ezen a fájlon írási joggal. Ne feledjük azonban, hogy biff bekapcsolta a szülőkönyvtár ***sticky*** bitjét. Ez az oka annak, hogy `crash` kísérlete kudarcba fullad, ahogy azt a *2. listán* láthatjuk is. Még egy megjegyzés a ***sticky*** bittel kapcsolatban: a beállításnak csak az adott könyvtárszinten érvényesül a hatása. Talán észrevettük, hogy az 1. lista az `extreme_casseroles` könyvtárban lévő két élmelyítő recept mellett egy `src` nevű könyvtárat is tartalmaz. Az `src` könyvtár tartalmára nem lesz hatással az `extreme_casserole` sticky bitjének beállítása, habár magára a könyvtárra igen. Ha `biff` az `src` könyvtár tartalmát is meg szeretné óvni a csoport tagjai által végrehajtott törléstől, az `src` könyvtár saját ***sticky*** bitjét is be kell állítania.

1. lista Az extreme_casseroles könyvtár tartalma

```
drwxrwxr-T   3 biff drummers   192 2004-08-10 23:39 .
drwxr-xr-x   3 biff drummers  4008 2004-08-10 23:39 ..
-rw-rw-r--   1 biff drummers    18 2004-07-08 07:40 chocolate_turkey_casserole.txt
-rw-rw-r--   1 biff drummers    12 2004-08-08 15:10 pineapple_mushroom_surprise.txt
drwxr-xr-x   2 biff drummers    80 2004-08-10 23:28 src
```

2. lista Törlési kísérlet a sticky bit bekapcsolt állapotában

```
crash> rm pineapple_mushroom_surprise.txt
rm: cannot remove
➥`pineapple_mushroom_surprise.txt´:
Operation not permitted
```

A setuid és setgid beállításai

Elérkeztünk a ***UNIX*** világának legveszélyesebb jogosultságbitjeihez, a ***setuid*** és ***setgid*** bitekhez. Egy futtatható bináris fájl esetén a ***setuid*** bit beállítása azt eredményezi, hogy a program a futtató felhasználótól függetlenül úgy fut, mintha a tulajdonosa futtatná. Hasonlóan, a ***setgid*** bitet beállítva egy futtatható fájl esetén az úgy fog futni, mintha a tulajdonos csoport tagja futtatná, függetlenül futtató személytől. Amikor azt a kifejezést használom, hogy úgy fut, azon azt értem, hogy ugyanazokkal a jogosultságokkal fut a program. Tegyük fel, hogy `biff` megír és lefordít egy olyan `killpms` nevű *C* programot, amelynek a hatása megegyezik az

```
rm /extreme_casseroles/
➥pineapple_mushroom_surprise.txt
```

parancséval, továbbá a

```
chmod +s ./killpms
```

paranccsal bekapcsolja a fájl ***setuid*** bitjét és beállítja a csoport-futtathatóságát is. A ***killpms*** fájl részletes jellemzőit kiíratva az alábbi sort láthatnánk:

```
-rwsr-xr--   1 biff drummers    22 2004-08-11 23:01
➥killpms
```

Ha `crash` lefuttatja ezt a programot, végül is sikerül elérnie, hogy törölje a ***Pineapple-Mushroom Surprise*** receptet – a `killpms` ugyanis olyan jogosultságokkal fut, mintha `biff` futtatta volna. Amikor a `killpms` megpróbálja letörölni a `pineapple_mushroom_surprise.txt` fájlt, sikerrel is jár, hiszen a fájl a tulajdonosa számára írható, a killpms pedig olyan jogosultságokkal rendelkezik, mint a tulajdonosa, `biff`. Ha egy programot a ***setuid*** bit beállításával akarunk futtatni, annak futtathatónak kell lennie a csoport és egyéb felhasználók számára – remélem nem kell magyaráznom, hogy miért. További jellemző, hogy a ***Linux*** rendszermagja a héjprogramok esetében nem veszi figyelembe a ***setuid*** és ***setgid*** biteket, ezek csak bináris (lefordított) futtatható fájlok esetében működnek.

A ***setgid*** ugyanúgy működik, csak a csoport-jogosultságokat használja. Ha egy futtatható fájl esetén a `chmod g+s filenév` paranccsal beállítjuk a ***setgid*** bitet és a fájl futtatható az egyéb felhasználók számára (`-r-xr-sr-x`), akkor a fájl futtatásakor a csoportazonosítót (***group ID***) használja a futtató felhasználóé helyett.

A fenti példa esetén, ha a `killpms` egyéb felhasználói jogosultságait r-x beállításra változtatjuk meg (`chmod o+x killpms`), és ugyanakkor beállítjuk a ***setgid*** bitet is (`chmod g+s killpms`), akkor függetlenül attól, hogy ki futtatja a `killpms`-t az mindenképpen a `drummers` csoport jogosultságaival fog rendelkezni, hiszen a `drummers` a fájl csoport-tulajdonosa.

A setgid és a könyvtárak

De mi a helyzet a könyvtárakkal? Nos a ***setuid*** nincs hatással a könyvtárakra, ellenben a ***setgid*** igen, ami nem teljesen magától értetődő tény. Rendes körülmények közt, ha létrehozunk egy fájlt, annak tulajdonosaként önműködően a saját felhasználói és (elsődleges) csoport-azonosítónk kerül beállításra. Például ha biff létrehoz egy fájlt, annak tulajdonosa `biff` lesz, a csoport-tulajdonosa pedig a `drummers` csoport, amennyiben a `drummers` `biff` elsődleges csoportja az */etc/passwd* lista alapján.

Beállítva azonban egy könyvtár ***setgid*** bitjét, az adott könyvtárban létrehozott fájlok a könyvtár csoport-tulajdonosát fogják örökölni. Ez akkor hasznos, ha a rendszerünk felhasználói jellemzően másodlagos csoportoknak is tagjai és rendszeresen hoznak létre olyan fájlokat, amelyeknek e csoportok többi tagja számára is elérhetőknek kell lenniük. Például ha az `animal` nevű felhasználó az */etc/group* fájlban úgy szerepel, mint a `drummers` csoport másodlagos tagja, de az `/etc/passwd` listában az elsődleges csoportja a `muppets`, akkor `animal` minden gond nélkül létrehozhat az `extreme_casseroles/` könyvtárban fájlokat `drwxrwx--T` jogosultság-beállítással. Mivel alapértelmezésben `animal` fájljai a `muppets` csoporthoz tartoznak, nem pedig a `drummers` csoporthoz, a `drummers` csoport többi tagjának sem olvasási, sem írási joga nem lesz `animal` receptjein, amíg `animal` kézzel át nem állítja a fájljai csoport-tulajdonosát (`chgrp drummers újfájl`) vagy meg nem változtatja az egyéb felhasználókra vonatkozó jogosultságokat (`chmod o+rw újfájl`).

Másrészről viszont ha `biff` vagy a rendszergazda beállítja az `extreme_casseroles/` könyvtár ***setgid*** bitjét (`chmod g+s extreme_casseroles`), akkor `animal` itt létrehozott új fájljainak csoport-tulajdonosa a `drummers` lesz a könyvtár tulajdonos csoportjának megfelelően. Minden egyéb jogosultság változatlan marad. Ha a kérdéses könyvtár már eleve nem írható a csoport által, akkor a `setgid` bitnek semmilyen hatása nincs, hiszen a csoporttagok nem hozhatnak benne létre fájlokat.

Ezzel megismertük az összes jogosultsági lehetőséget: az olvasási, írási, futtatási jogokat, a ***sticky***-bitet, a ***setuid*** és

3. lista Az alapértelmezett jogosultságmaszkunk ellenőrzése

```
mick@localhost:/home/mick> umask
0022
```

4. lista Egy 0022-es maszkkal létrehozott fájl és könyvtár

```
-rwxr-xr-x 2 mick users 48 2004-08-13 08:31 default_dir
-rw-r--r-- 1 mick users  4 2004-08-13 08:31 default_file
```

setgid biteket. Ha mind a hatot megértettük, valószínűleg a ***Linux*** felhasználók kisebbik feléhez tartozunk. De ezzel még nincs vége!

Numerikus módszerek

Eddig kódokat használtunk a jogosultságok jelölésére: r jelentette az olvasási jogot, w az írásit és így tovább. Mondanom sem kell, hogy a minden mással együtt a rendszerünk a jogosultságokat is számokként tárolja. A `chmod` felismeri mind a kódokkal (`u+rwx,go-w`), mind pedig a számokkal megadott jogosultság-módosítókat.

Egy számkód négy számjegyből áll, amelyek balról jobbra haladva a különleges jogokat, tulajdonosi jogokat, csoportjogokat és az egyéb jogokat jelentik. Emlékezzünk rá, hogy az egyéb jogok azokra a felhasználókra vonatkoznak, akik nem tartoznak sem a tulajdonos, sem a csoporttagok kategóriájába. Például a 0700 számkód szerint nincs különleges jogosultság, minden tulajdonosi jog be van kapcsolva, és semmilyen csoport- vagy egyéb jogosultság sincs.

Minden jogosultság-típushoz egy meghatározott számérték tartozik, az egyes jogosultság-típusokhoz tartozó értékek pedig az egyes helyiértékeken összeadódnak: a számjegy az összes beállítani kíván bit összegét jelenti. Ha például a tulajdonosi jogosultságok értéke 7, ez a 4-nek (az olvasási jog értéke), a 2-nek (az írási jog értéke) és az 1-nek (a futtatási jog értéke) összegeként áll elő.

Ahogy az előbb említettem, az egyes jogokhoz tartozó értékek: 4–olvasás, 2–írás és 1–futtatás. (Én úgy jegyeztem meg ezeket, hogy ismételgettem magamban az „olvasás-írás-futtatás, 4-2-1” mondókát.) Miért nem 3, kérdezhetné valaki? Azért, mert így biztosítható, hogy ne legyen két olyan jogosultság-kombináció, amihez ugyanaz az érték tartozik.

A különleges jogosultságok a következők: 4 jelenti a ***setuid*** bitet, 2 a ***setgid*** bitet, az 1 pedig a ***sticky*** bitet. Például a 3000 a ***setgid*** és ***sticky*** bit bekapcsolását jelenti, miközben semmilyen más jogosultság nincs – ennek a beállításnak nincs persze sok értelme.

Nézzünk még egy példát a numerikus megadásra. Ha futtatom a `chmod 0644 mycoolfile` parancsot, az *1. ábrán* látható jogosultságokat állítom be a `mycoolfile` fájl számára.

A numerikus módszer teljesebb leírását a `coreutils` parancs ***Numeric Modes*** részében találhatjuk meg, vagyis ha kiadjuk az `info coreutils numeric` parancsot.

Az umask parancs

Mielőtt más témába kezdenék, szeretném bemutatni az utolsó jogosultsággal kapcsolatos parancsot. Az `umask` a ***Bash*** héjba beépített parancs, amely kiírja vagy beállítja az alapértelmezett jogosultságmaszkunkat. A sajátunkat az umask parancs paraméterek nélküli beírásával nézhetjük meg, ez egy négyjegyű számot fog eredményül adni.

A rendszeremen a 2. listának megfelelő választ kaptam.

A 0022 jelentése a következő: nincs különleges jogosultság, nincs tulajdonosi jogosultság, a csoport- és egyéb jogosultságok közül pedig az írási van bekapcsolva. Ez meg hogy lehetséges?

A válasz, hogy valójában az `umask` nem a kóddal, hanem maszkokkal dolgozik. A 0022 az a szám, amit a 0777 értékből kivonva dönti el a rendszer, hogy a létrehozott fájlokhoz milyen jogosultságokat rendel: `0777-0022 = 0755`.

Aha! Szóval a létrehozott fájljaim tulajdonosa rendelkezni fog olvasási, írási és futtatási jogokkal (7 = 4+2+1), a csoport- és egyéb jogosultságok közül pedig az olvasás és futtatás lesz beállítva (5 = 4+1). Igaz ez? Majdnem. Valójában az `umask` a futtatási jogosultságot önműködően csak a könyvtárakra állítja be. Még ha a jogosultság-maszkunk szerint szerepel is a futtatási jog, a létrehozott közönséges fájlok futtatás-bitje nem kerül önműködően bekapcsolásra.

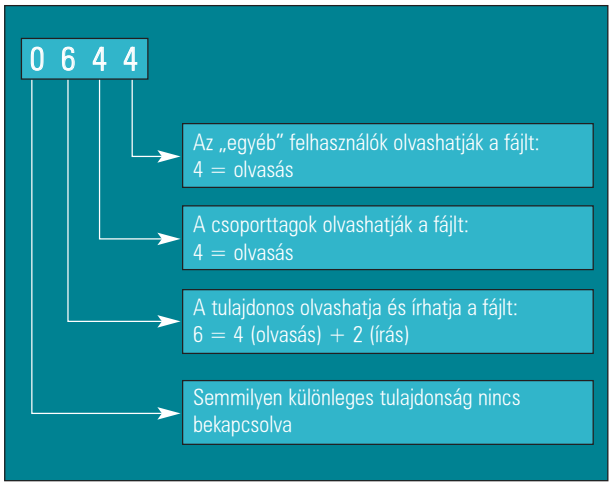
Tehát a 0022 jogosultságmaszkommal, amely az alapértelmezett jogokat 0755 értékre állítja be, a `default_dir` könyvtárban létrehozva egy `default_file` nevű fájlt, a két tételnek a részletes adatai a *4. listán* lévő képet fogják mutatni.

Az alapértelmezett jogosultságmaszkunk megváltoztatásához egyszerűen futtassuk az `umask` parancsot, paraméterként megadva a kívánt maszkot. Például ha azt szeretném, hogy az összes fájlom rendelkezzen csoport-olvasási joggal, de semmilyen más jogot nem szeretnék beállítani, ez a 0740 numerikus kóddal fejezhető ki. Ezt kivonva a 0777 kódból a 0037 maszkot kapom, vagyis az alapértelmezett maszkom beállítására az `umask 0037` parancsot kell használnom. Ez az új maszk azonban csak az aktuális munkafolyamatra és az ebből indított héjakra érvényes, ha álladósítani szeretném a beállítást, a ***.bashrc*** fájlomba új sorként be kellene írnom az `umask 0037` parancsot.

A su és sudo parancsok

Most, hogy már mindent tudunk a jogosultságokról, numerikus kódokról és a maszkokról, szólnom kell néhány szót arról, hogy mi a helyzet a felhasználókkal, csoportokkal és ezek jogaival a gyakorlatban. A ***UNIX*** biztonsági rendszerével nagyon gyakran az a gond, hogy egy adott rendszeren a hatókörök nagyjából úgy néznek ki, hogy a root mindent megtehet, a felhasználók meg szinte semmit.

Sajnos sokkal könnyebb kiadni egy gyors `su` parancsot, és ezzel egy időre megszerezni a root-jogosultságokat, mint létrehozni egy jól meghatározott jogosultságokkal rendelkező csoportrendszert, amelyben a rendszergazdák és az alrendszerek felügyelői pontosan azokkal a jogosultságokkal rendelkeznek, amelyekre szükségük van. Használhatjuk persze a `su` parancsot a `-c` kapcsolóval is, ami csak egy bizonyos parancs root-ként való futtatását teszi lehetővé az egész héj helyett (például `su -c rm fájlnév.txt`), de ehhez szükség van a root-jelszó beírására. Az pedig soha nem jó,



1. ábra A mycoolfile jogosultságainak magyarázata

FONTOS FIGYELMEZTETÉS!

A ***setuid*** és ***setgid*** bitek nagyon veszélyesek lehetnek, ha root vagy valamilyen kiemelt jogosultságokkal rendelkező felhasználó vagy csoport tulajdonában lévő fájlokon kerül beállításra. Azért mutatom be a ***setuid*** és ***setgid*** működését, hogy megértsük, mire való – nem gondolom, hogy a gyakorlatban valami fontos felhasználási célja lenne számotokra. A **sudo** parancs, amit a cikk további részében ismertetek majd, sokkal alkalmasabb eszköz arra, hogy valakit root-jogosultságokkal ruházzunk fel.

ha néhány személyen kívül más is ismeri a root-jelszót. A root-mindent-visz probléma megoldásának egy másik megközelítési módja a ***SELinux***-hoz hasonló szerep-alapú hozzáférési rendszer (***RBAC***) használata, amely olyan hozzáférési rendszert léptet hatályba, ami csökkenti a root hatáskörét. Ez azonban talán még bonyolultabbá teszi a helyzetet, mint a megfelelő csoportok és csoport-jogosultságok beállítása, amivel nem akarom azt mondani, hogy a ***SELinux*** és társai ne lennének nagyon hasznosak – én nagyon szeretem az ***RBAC***-t.

Egy jó középutat jelenthet a `sudo` parancs használata. A `sudo` lehetővé teszi a felhasználók számára, hogy egyes parancsokat root jogosultsággal futtassanak anélkül, hogy ismerniük kellene a root-jelszót. A `sudo` mára a legtöbb Linux rendszercsomag alapelemét képezi.

A `sudo` beállításait az */etc/**sudoers*** fájl tartalmazza, de ennek közvetlen szerkesztésére nincs szükség. Ehelyett kiadhatjuk a `visudo` parancsot, amely megnyitja a fájl szerkesztőjét, ami alapesetben a `vi`. Az `EDITOR` környezeti változó megváltoztatásával más karakteres szerkesztőprogramot is beállíthatunk. Például a */**usr**/**bin**/**gedit*** használatához az alábbi parancsot kell kiadnunk:

```
export EDITOR=/usr/bin/gedit
```

Hely hiányában nem tudom részletezni a ***sudoers*** fájl szerkezetét, a ***sudoers(5)***, ***sudo(8)*** és ***visudo(8)*** súgóoldalakon olvashatjuk el a teljes leírást. Nézzünk inkább egy gyors példát.

Emlékszünk még a `crash` felhasználó próbálkozására, hogy megszabaduljon a ***Pineapple-Mushroom Surprise*** recepttől? Bár ebben az esetben nem kellene ilyen durva eszközökhöz nyúlnunk – a bemutatott jogosultság-technikák megfelelő eszközt biztosítanak ehhez –, használhatjuk a `sudo` parancsot arra, hogy `crash` elérje célját, feltéve, hogy `biff` rendelkezik root-jogosultsággal. Először is váljunk root felhasználóvá (`su -`). Ezután futtassuk a `visudo` parancsot. Ennek hatására a `vi` szerkesztőprogramba kerülünk, amelyben az */**etc**/**sudoers*** fájl van épp megnyitva (ha a `vi` használatában újoncok vagyunk, a ***vi(1)*** súgóoldalt érdemes elolvasnunk). Menjünk el a fájl végéig, majd adjuk az alábbi sort a fájlhoz:

```
crash localhost=/bin/rm /home/biff/
➥extreme_casseroles/pineapple_mushroom_surprise.txt
```

Mentsük a fájlt, majd lépjünk ki. Most `crash` a feladat elvégzéséhez beírja a következő parancsot:

```
sudo rm /home/biff/extreme_casseroles/
➥pineapple_mushroom_surprise.txt
```

A parancs a felhasználó jelszavának beírását kéri. Miután hibátlanul beírta, az alábbi parancs fut le, mégpedig root felhasználóként:

```
/bin/rm /home/biff/extreme_casseroles/
➥pineapple_mushroom_surprise.txt
```

Ezzel megtörtént a kérdéses fájl törlése. Egy másik megoldás, ha az */**etc**/**sudoers*** fájlban lévő sor a következőképpen fest:

```
crash localhost=/bin/rm /home/biff/
➥extreme_casseroles/*
```

Ebben az esetben `crash` bármit letörölhet az `extreme_casseroles` könyvtárban függetlenül a ***sticky*** bit beállításától.

A `sudo` nagyon kényelmes eszköz, de legalább ilyen veszélyes is lehet, bánjunk vele körültekintően – a root jogosultságaival nem szabad játszadozni. Tényleg okosabb, ha a felhasználói és csoportjogosultságokat állítjuk be a célnak megfelelően, mint ha kiadjuk a root jogosultságait, még ha csak a `sudo` parancson keresztül is. Még az is jobb, ha egy olyan ***RBAC***-alapú rendszert használunk, mint a ***SELinux***, amennyiben rendszer védelme ezt megkívánja. Ez alkalommal ennyi fért bele. Remélem hasznos volt ez az ismertető. Legyetek óvatosak a következő alkalomig!

Linux Journal 2004. november, 127. 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.

Cikkgyűjtés RSS használatával

A hírgyűjtő rendszerek nem csak weblogok figyelemmel kísérésére alkalmasak. Az RSS segítségével a látogatókat könnyen tájékoztathatjuk egy-egy webhely vagy alkalmazás változásairól is.

Amikor elkezdtem használni a webet, még az volt a szokás, hogy minden új weboldal fenntartója küldött egy levelet ***Tim Berners-Leenek***, amiben megadta az ***URL***-t és az oldal témájának rövid leírását. ***Tim*** válaszában röviden összefoglalta személyes megjegyzéseit, majd frissítette a weboldalakat tartalmazó főlistát, amit bárki szabadon megnézhetett. A webes közösség aktív résztvevői rendszeresen átnézték a listát – majd utódját is, amelyet mellesleg a ***Mosaic*** böngésző készítői állítottak össze –, és kimazsolázták belőle az új vagy frissített oldalakat, nehogy lemaradjanak valami érdekesről.

A web alig egy évtized alatt túl nagyra nőtt ahhoz, hogy az új webhelyek listáját kézi munkával fenn lehetne tartani. Még ha találnánk is rá embereket, a látogatók csak egy kis töredékét tudnák befogadni a minden egyes nap nyilvánosságra kerülő új tartalomnak. Ha figyelembe vesszük, hogy napjainkban már több százezer weblog, röviden blog üzemel, és sokat közülük elég gyakran frissítenek, akkor nyilvánvalóvá válik, hogy a feladat rendkívül nehezen oldható meg.

Az egyik megoldás, hogy böngészőnkben könyvjelzőket használunk, ám ezek rendszeres végiglátogatása rendkívül vesződséges, főleg, ha naponta többször kívánjuk elvégezni. Milyen jó lenne, ha mindegyik oldal maga jelezné változásait, így csak akkor kellene meglátogatnunk őket, amikor valóban érdemes!

Az ötlet persze nem új, a tartalmukat hirdető weboldalak elképzelése már évekkel ezelőtt megszületett. Sajnos be kell vallanom, csak néhány hónappal ezelőtt jöttem rá, hogy mennyire elmaradott módszert alkalmazok, amikor könyvjelzők alapján látogatom végig a kedvenc oldalaimat. Egy ***RSS*** összesítő segítségével – vagyis egy olyan programmal, amely összegyűjti a különféle helyek ***RSS*** cikkeit, és jelzi, ha valahol frissítettek – mindezt sokkal kevesebb idő alatt el tudom végezni.

Ebben a hónapban a népszerű ***RSS*** (***Really Simple Syndication; valóban egyszerű cikkgyűjtés***) vagy ***RDF Site Summary*** (***RDF webhely-összegzés***) formátumcsaládot tárgyaljuk, megvizsgáljuk, hogy tagjai milyen célokra használhatók, illetve a szabványoknak megfelelő cikkek hogyan állíthatók elő.

Egyszerű RSS

Az ***RSS*** tulajdonképpen a ***Netscape*** gyermekének mondható – az internetes céget azóta az ***AOL*** felvásárolta, gyakorlatilag eltüntette. A ***Netscape*** ötlete az volt, hogy a felhasználóknak több forrásból származó híreket kínál egyetlen oldalon. Ebből a célból született meg az ***RSS 0.90***. Aki a ***Netscape*** portálján keresztül híreket szeretett volna közzétenni, ***RSS*** segítségével tehette ezt meg. A ***Netscape*** rendszere lekérte a megfelelő ***RSS*** dokumentumot a kérdéses webhelyről, majd közzétette a kapott anyagot.

Bár az ***RSS 0.90*** kisebb forradalmat indított el, rendkívül bonyolult volt. ***Dave Winer***, a ***Userland Software*** későbbi vezetője az ***RSS***-t sokkal egyszerűbb szabálygyűjteménnyé dolgozta át, nevét ***RSS 0.91***-re változtatta, majd elkezdte saját weblogjában, a ***scripting.com*** oldalon népszerűsíteni.

Az ***RSS 0.91*** szinte azonnal megjelent a web összes szegletében, és ***Dave*** narancsszínű ***XML*** gombjai, amelyek az egyes helyek ***RSS***-képességét jelezték, hatalmas népszerűségre tettek szert. Néhány év alatt további ***RSS***-változatok is megjelentek. Az ***RSS 1.0*** fejlesztését egy webes csoportosulás végezte, míg az ***RSS 2.0*** – ***Dave*** vezényletével – a ***0.9x*** továbbfejlesztéseként jelent meg.

Aki jól figyelt, észrevette, hogy jelenleg három különböző, ám egyaránt ***RSS***-nek nevezett cikkgyűjtő formátum is létezik. Bár vannak közöttük hasonlóságok, a különféle változatok erősen eltérő formátumokat adnak meg.

Az ***RSS*** sok tekintetben hasonlít a ***HTML***-re és a ***HTTP***-re, és eleinte egy kisebb csoport által fejlesztett, könnyen megérthető és könnyen megvalósítható szabványról volt szó. Az elmúlt évek során azonban mindhárom szabvány hatalmas fejlődésen ment keresztül, ami rugalmasságuk és egyszerűségük részleges elvesztéséhez vezetett.

A legegyszerűbb – és nem kevéssé népszerű – változat az ***RSS 0.91***. Minden tartalmat `<rss>` címkék közé kell helyezni, ez tartalmazza a változat megjelölését, és egy darab `<channel>` (csatorna) elem szerepelhet benne. A kötelező címkéket `title`, `link`, `description`, `language` és `image` (rendre cím, hivatkozás, leírás, nyelv és kép) egy vagy több `<item>` (tétel) elem követi. Minden tétel saját címmel, hivatkozással és leírással rendelkezik. Példaként lássunk egy egyszerű ***RSS***-cikket saját weblogomból (*1. lista*).

1. lista

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE rss PUBLIC
"-//Netscape Communications//DTD RSS 0.91//EN"
"http://my.netscape.com/publish/formats/rss-
0.91.dtd">

<rss version="0.91">

<channel>
  <title>Altneuland</title>
  <link>http://altneuland.lerner.co.il/</link>
  <description>Reuven´s Weblog</description>
  <item>
    <title>Independence Day</title>
     ➥<link>http://altneuland.lerner.co.il//
     ➥40</link>
  </item>
  <item>
    <title>Linux desktops for the masses?
     ➥Ha!</title>
     ➥<link>http://altneuland.lerner.co.il//
     ➥39</link>
  </item>
</channel>
</rss>
```

Ha megvizsgáljuk a fenti *RSS*-cikket, láthatjuk, hogy a korábban említett ***RSS 0.91*** szabálygyűjtemény előírásainak nem felel meg, ugyanis hiányzik belőle a kötelező nyelv és kép elem, illetve nem mindegyik tételhez tartozik leírás. Rutinosabbaknak ez aligha okoz meglepetést, régebben a *HTML* esetében is láthattunk ilyesmit: a programok készítői megelégednek olyan kimenet előállításával, amely hiányos ugyan, ám az alkalmazási területek túlnyomó részén megállja a helyét. Ezt a divatot követi a ***COREBlog*** is (jelenleg ezt használom saját weblogom készítéséhez), segítségével használható, ám az ***RSS 0.91***-nek csak részben megfelelő cikkeket állíthatunk elő.

A cikkek létrehozása

Ha szabványos *RSS*-cikket szeretnénk előállítani, próbálkozzunk meg a népszerűbb nyelvek mindegyikéhez elérhető nyílt forrású modulok valamelyikével. A Perl-hívők számára például az `XML::RSS` modul jelent segítséget, ezt bármelyik ***CPAN*** tükörről le lehet tölteni (lásd az internetes források részt).
Ha a modul segítségével *RSS*-cikket szeretnénk létrehozni, az alábbihoz hasonló egyszerű programot kell írnunk.

```
#!/usr/bin/perl

use strict;
use diagnostics;
use warnings;

use XML::RSS;

my $url = "http://altneuland.lerner.co.il/";

my $rss = new XML::RSS (version => ´0.91´);
$rss->channel(title    => ´Altneuland´,
       link            => $url,
       language        => ´en´,
       description     => "Reuven Lerner´s
                          ➥Weblog");

$rss->add_item(title => ´Being scared´,
        link         => "$url/43/index_html",
        description  => ´Blog entry´
       );

print $rss->as_string;
```

A program első lépéseként egy új `XML::RSS` objektumot hozunk létre, egyúttal azt is kinyilvánítjuk, hogy az ***RSS*** szabvány ***0.91***-es változatát kívánjuk használni. Ezután az egyedi tételeket adjuk meg. Az `image` címkét elhagyhatjuk. Bár az `XML::RSS` modul a csatorna leírásából bármelyik címke elhagyását lehetővé teszi, például a cím és a hivatkozás kihagyásával az egész cikk értelmét veszti.
Következő lépésünk a csatorna feltöltése az egyes tételekkel. Amikor ezzel végeztünk, készen állunk az *RSS* kimenet előállítására, amely a következőképpen fog alakulni:

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE rss PUBLIC
"-//Netscape Communications//DTD RSS 0.91//EN"
"http://my.netscape.com/publish/formats/rss-
0.91.dtd">

<rss version="0.91">

<channel>
<title>Altneuland</title>
<link>http://altneuland.lerner.co.il/</link>
<description>Reuven Lerner´s Weblog </description>
<language>en</language>

<item>
<title>Being scared</title>
<link>http://altneuland.lerner.co.il/43/index_html
➥</link>
<description>Blog entry</description>
</item>

</channel>
</rss>
```

Az *RSS*-cikkek előállítására szolgáló programok nagy része nem fogja ismételten meghívni az `$rss->add_item()` függvényt, ahogy én tettem. Ha weblog, kereskedelmi hírlap vagy egyéb gyakran frissített oldal változásait szeretnénk jelezni, akkor célszerűbb egy olyan *RSS*-cikket készítenünk, amely egy könyvtár fájljain vagy – még jobb megoldás – egy relációs adatbázis sorain halad végig újra és újra.

Az alábbi kódrészlet például az utolsó 24 órában megjelent weblog bejegyzéseket gyűjti össze egy *PostgreSQL* alatti, `weblog_entries` nevű táblába.

```
# Az összes az utolsó 24 órában készült bejegyzés
# kigyűjtése
my $sql = "SELECT entry_id, title, link,
description
          FROM weblog_entries
          WHERE when_entered >= (NOW() - interval
                              ➥'1 day')";

# Az SQL-utasítás összeállítása
my $sth = $dbh->prepare($sql);

# Az SQL-utasítás végrehajtása
my $result = $sth->execute;

# Végiglépkedés a kapott sorokon
while (my $rowref = $sth->fetchrow_arrayref)
{
my ($id, $title, $link, $description) = @$rowref;

$rss->add_item(title => $title,
            link => $link,
            description => $description
           );
}
```

A fentiekből azonnal nyilvánvalóvá válik az is, hogy miért érdemes relációs adatbázisban tárolni a weblogokat. Ha a bejegyzések bekerülnek valamilyen adatbázisba, az új szolgáltatások, például a cikkgyűjtés megvalósítása már egyszerű. Noha az `XML::RSS` biztosít lehetőséget a begyűjtött cikkek számának korlátozására (erre példakódot is találunk), erre a feladatra az adatbázisok sokkal alkalmasabbaknak tűnnek, hiszen esetükben egyszerűen a `LIMIT` módosítóval be tudjuk állítani a kapott sorok számának felső határát.

Áttérés az RSS 1.0 változatra

Az ***RSS 1.0*** egyfajta válasz volt az ***RSS 0.91***-re, célja a közelítés volt a ***World Wide Web Consortium (W3C)*** különféle szabványaihoz, többek közt az ***RDF***-hez. A változatszám alapján azt hihetnénk, hogy az ***1.0*** a ***0.91*** frissítése, ám a jelölés rendkívül szerencsétlen, hiszen két teljesen független megoldásról van szó. A ***0.91*** (és utódja, az ***RSS 2.0***) fejlesztését a közösség visszajelzései alapján ***Dave Winer*** végezte, az ***1.0*** változat viszont fejlesztők egy nyitott közösségének munkája nyomán állt elő. Az ***RSS 0.91*** és ***2.0*** között több hasonlóságot fedezhetünk fel, mint az ***1.0*** és a másik két változat bármelyike között, ami nem meglepő módon számos félreértés forrása.

Az ***RDF*** (***Resource Development Framework***, erőforrás-fejlesztési keretrendszer) a ***W3C*** fejlesztése, a szemantikai, jelentéstani web létrehozására irányuló tervezet része, amelynek célja az, hogy a webet az embereken túl a számítógépek számára is érthetővé tegye. Ehhez alapfeltétel a metaadatok, vagyis a webhelyek által átadott anyagokat kísérő, a felhasználók számára láthatatlan leírók szabványosítása. Az ***RDF*** is egy próbálkozás erre a szabványosításra. Az ***RSS 1.0*** tehát a cikkgyűjtést az ***RDF***-hez csatolja, gondoskodva az ***XML*** névterek használatáról is. Az ***XML*** névterek segítségével különböző ***XML***-megadásokat is össze tudunk fogni egyetlen dokumentumba.

Ha az ***RSS 1.0***-nak megfelelő cikket szeretnénk összeállítani, akkor a fenti programban egyetlen apró módosítást kell végrehajtanunk, mégpedig át kell írnunk az `XML::RSS` esetében megjelölt változatszámot. Lásd az *1. listát*.

2. lista

```
my $rss = new XML::RSS (version => '1.00');
A módosítást követően a létrejövő RSS-cikk picit
eltérően épül majd fel.
<?xml version="1.0" encoding="UTF-8"?>

<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/
          ➥22-rdf-syntax-ns#"
 xmlns="http://purl.org/rss/1.0/"

 xmlns:taxo="http://purl.org/rss/1.0/modules/
          ➥taxonomy/"
 xmlns:dc="http://purl.org/dc/elements/1.1/"

 xmlns:syn="http://purl.org/rss/1.0/modules/
          ➥syndication/"
 xmlns:admin="http://webns.net/mvcb/"
>

<channel
rdf:about="http://altneuland.lerner.co.il/">
<title>Altneuland</title>
<link>http://altneuland.lerner.co.il/</link>
<description>Reuven Lerner's Weblog
</description>
<dc:language>en</dc:language>
<items>
 <rdf:Seq>
  <rdf:li rdf:resource=
     "http://altneuland.lerner.co.il/
     ➥43/index_html" />
 </rdf:Seq>
</items>
</channel>

<item rdf:about=
 "http://altneuland.lerner.co.il/43/index_html">
<title>Being scared</title>
<link>http://altneuland.lerner.co.il/43/
      ➥index_html</link>
<description>Blog entry</description>
</item>

</rdf:RDF>
```

3. lista

```
<?xml version="1.0" encoding="UTF-8"?>

<rss version="2.0"
 xmlns:blogChannel="http://backend.userland.com/
                    ➥blogChannelModule">

<channel>
<title>Altneuland</title>
<link>http://altneuland.lerner.co.il/</link>
<description>Reuven Lerner´s Weblog
</description>
<language>en</language>

<item>
<title>Being scared</title>
<link>http://altneuland.lerner.co.il/43/
      ➥index_html</link>
<description>Blog entry</description>
</item>

</channel>
</rss>
```

A kimenetben több említésre méltó elem is található. Érdemes például észrevenni, hogy benne számos névteret adunk meg és használunk, ezek bevezetése az `xmlns` jellemzőkkel történik, de további, kifejezetten az *RDF*-hez kötődő jellemzőket is használunk, mint az `rdf:about` és az `rdf:resource`.
A fentiekben az ***RSS 1.0*** által kínált lehetőségeket lényegében kihasználatlanul hagyjuk, hiszen a szabvány nagy számú beállítás megadására kínál módot. Például, az `$rss->channel()` híváshoz egy `syn` részt adva beállíthatjuk oldalunk cikkgyűjtési frissítésének gyakoriságát.
Az ***RSS 1.0*** a ***Dublin Core***-t is támogatja, ez egy folyamatosan növekvő népszerűségű szabványos megoldás a dokumentumok címkézésére.

RSS 2.0

A jó hír az, hogy – amint láttuk – az ***RSS 1.0*** formátumú cikkek előállítása vagy feldolgozása nem különösebben nehezebb, mint az ***RSS 0.91*** formátumúaké, feltéve persze, hogy naprakész eszközökkel rendelkezünk. Csakhogy az ***RSS 1.0*** viszonylag bonyolult, néhányan úgy vélik, hogy túlzottan is.
Mivel a számos próbálkozás ellenére nem sikerült megegyezésre jutni az ***RSS 1.0*** tekintetében, fejlesztők egy csoportja Atom név alatt új tervezetet indított. Az Atomról most nem szólnék, azt viszont érdemes tudni róla, hogy felbukkanása ösztönözte a ***Winer*** által vezetett ***RSS*** tábort az ***RSS 2.0*** kifejlesztésére.
Ha ***RSS 2.0***-megfelelő cikkeket szeretnénk előállítani, akkor újfent a kívánt változatszámot kell módosítanunk:

```
my $rss = new XML::RSS (version => ´2.0´);
```

Ügyeljünk arra, hogy a változatszám ***2.0***; nem ***2*** és nem ***2.00***. Utóbbiak egyike sem fog működni, a változatszám ellenőrzése ugyanis karakterláncok összevetésével és nem számértékek összehasonlításával történik. Hogyan néz ki az ***RSS 2.0***? Nem fog túl sok meglepetést okozni (*3. lista*). Amit fent látunk, nagyon hasonlít a ***0.91***-es ***RSS***-re, de akár az ***RSS 1.0*** lecsupaszított változatának is tekinthető. Ha viszont felidézzük azt a tényt, hogy az ***RSS 2.0*** a ***0.91*** utódja, melynek feladata – a kis méret, az egyszerű megvalósíthatóság és rugalmasság megőrzése mellett – a felmerült hiányosságok pótlása volt, azonnal minden megvilágosodik.
Az ***RSS 2.0*** számos fejlesztést hordoz magában a ***0.91***-hez képest, a legfontosabb talán a névterek modulokként való használata, amivel új szolgáltatások valósíthatók meg.
Az ***RSS 2.0*** által megadott vagy használt névterek száma messze nem közelíti meg az ***1.0***-nál látottat, ám ennek fő oka az, hogy nem próbálkozik meg az ***RDF*** megvalósításával.
Winer részben a ***RSS 2.0*** szabálygyűjtemény szerzői jogának megtartása miatt őt ért kritikák hatására a jogokat a ***Harvard Universitynek*** adta át. Feltételezhető, hogy ***Winer*** továbbra is fontos szerepet fog játszani az ***RSS 2.0*** fejlesztésében, ám nem ő lesz az, aki végső soron dönt a használattal vagy a bővítésekkel kapcsolatos kérdésekben.
A szétválás mindettől függetlenül véglegesnek tűnik. Kialakult egy ***Atom csoport*** és egy ***RSS csoport***, én nem nagyon hiszem, hogy valaha is közös nevezőre jutnak. A fejlesztői táborok által kitűzött célokat szemlélve ez a legkevésbé sem meglepő – végül is nem várhatjuk el, hogy ugyanaz a szabálygyűjtemény egyszerre törekedjen az egyszerűségre és a rugalmasságra.

Összefoglalás

Ez alkalommal az ***RSS*** jelenleg is használatban lévő változatait tekintettük át, illetve összevetettük stílusukat és fejlesztőik célkitűzéseit. Szerencsére, ha valaki szeretne egyszerű, begyűjtésre alkalmas cikkeket készíteni, különösebb nehézségekre nem kell számítania. Bár a programozók az egyes változatokra egyedileg jellemző mezőket is hozzáadhatnak, az alap mindegyik ***RSS***-változatnál azonos, függetlenül attól, hogy az együttműködésnek még a szándéka is hiányzik. A létrejövő ***RSS***-cikkek természetesen egészen eltérő kinézetűek is lehetnek, függően a kiválasztott változattól.
Következő írásomban az ***RSS*** nemrég megjelent, ám egyre elterjedtebb vetélytársáról, az ***Atom*** cikkgyűjtő formátumról lesz szó. Ha azzal is végeztünk, akkor áttekintjük, hogyan készíthetünk saját cikkgyűjtőt, amellyel különböző forrásokból származó cikkeket tudunk értelmezni és kezelni. Megvizsgáljuk az ***RSS*** használatának különféle módjait is, illetve azt, hogy a cikkgyűjtők révén hogyan juthatunk hozzá a legfrissebb hírekhez és a legújabb véleményekhez.

Linux Journal 2004. október, 126. szám

Reuven M. Lerner (➲ http://www.lerner.co.il/atf) Nyílt forrású programokra, valamint web- és adatbázis-alkalmazásokra szakosodott tanácsadó. Könyve, a Core Perl, 2002 januárjában jelent meg a Prentice Hall gondozásában. Reuven feleségével és lányaival nemrég költözött Chicagóba.

Tudományos célú képalkotás a POV-Ray segítségével

Egy kis munkával a Persistence of Vision Raytracer (POV-Ray) felhasználható arra, hogy lebegőpontos tudományos adatfájlok alapján lélegzetelállító háromdimenziós ábrákat hozzunk létre.

Meteorológus vagyok a ***Michigani Központi Egyetemen***, ahol az ***Illinois Egyetemen*** dolgozó kollégáimmal a szupercellás zivatarok viselkedését kutatom. Ezek a hosszú életű örvénylő szörnyetegek minden tavasszal nagy felfordulást okoznak a ***Nagy Síkságokon*** az ***Egyesült Államokban***. E félelmetes viharok tanulmányozásának elsődleges eszköze számomra egy ***NCOMMAS*** nevű numerikus modell, egy ***FORTRAN 90*** nyelven írt számítógépes program, amely a fizikai egyenletek felhasználásával emulálja a légkör háromdimenziós időbeli állapotváltozását. Ez a modell egy négy órás vihar szimulációja során roppant mennyiségű adatot állít elő, amely még veszteséges tömörítéssel is 200 GB nagyságrendű marad. A kutatásaim során felmerült egyik nagy kihívás az volt, hogy módot találjak ezeknek az adatoknak olyan tudományos igényű megjelenítésére, amellyel betekintést nyerhetnénk a szimulált vihar fizikai természetébe.

A háromdimenziós adatok megjelenítésének egyik módja egy sugárkövető (***raytracer***) használata. A sugárkövető egy olyan számítógépes program, amely a pontkép előállításához a fény viselkedését szimulálja, amint az a háromdimenziós térben elhelyezett virtuális tárgyakkal kölcsönhatásba lép (*1. ábra*). Az így kapott bittérkép megjeleníthető ezután a számítógép képernyőjén, vagy lemezre menthető valamilyen ismert formátumban, mint amilyen a ***PPM*** vagy a ***TIFF***. A ***Persistence of Vision Raytracer***, vagy röviden ***POV-Ray*** egy népszerű nyílt forráskódú sugárkövető program, amely akkor keltette fel az érdeklődésemet, amikor a 90-es évek közepén a ***Wisconsin Egyetemen*** a doktori disszertációmat írtam. Abban az időben konvektív leáramlások – a viharfelhőkben időnként kialakuló lefelé irányuló légáramlások – háromdimenziós adatainak megjelenítésére kerestem megfelelő programot. Mivel a tudományos életben hozzá voltam szokva a megosztott forráskódokhoz és végzős diákként anyagilag sem álltam jól, olyan ingyenes és forráskódban terjesztett programot kerestem, amit letölthetek és különleges igényeimhez igazíthatok. A logikus választás akkor a ***POV-Ray*** volt és ma is megfelel az igényeimnek, amikor a kutatásaim adatait sugárkövetési módszerrel előállított képen szeretném megjeleníteni.

A tudományos adatok leképezése nem az a feladat, amelyre a ***POV-Ray*** eredetileg készült, és kevés kutató használja a programot erre a célra. Más sugárkövető programcsomagok a numerikus modellek terén jobban támogatják a kutatók munkáját, de ezek zárt forráskódúak és meglehetősen drágák. Ebben a cikkben körvonalazom azt a módszert, amellyel a ***POV-Ray*** képessé tehető a háromdimenziós tudományos adatok szintfelületeinek közvetlen megjelenítésére.

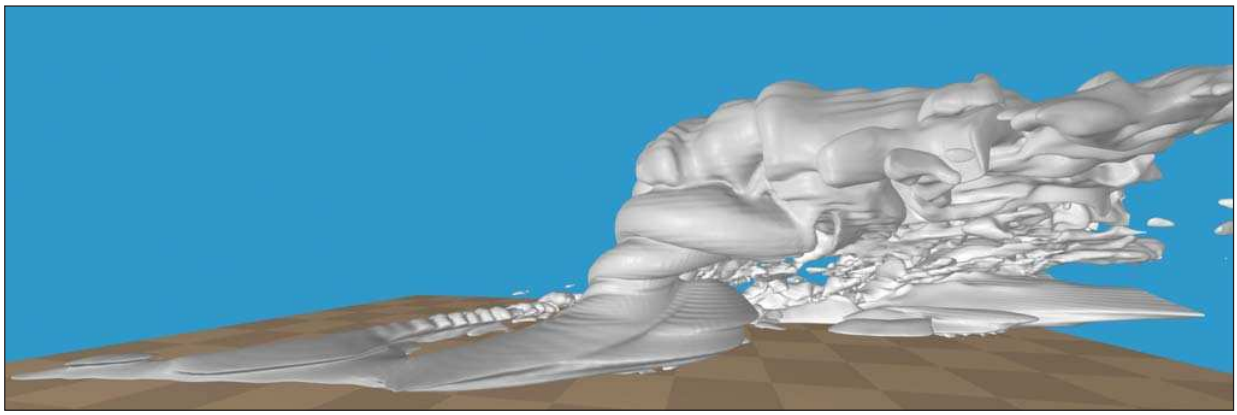
A forráskód megszerzése

Bár a ***POV-Ray*** bináris változatban is elérhető ***Linux***, ***Mac OS*** és ***Microsoft Windows*** operációs rendszerekhez, nekünk szükségünk lesz a forráskódra is ahhoz, hogy a foltokat alkalmazni tudjuk, és további módosításokat hajthassunk végre. A cikkben az írás idején elérhető legfrissebb, 3.5 változatot használom. A ***POV-Ray*** letöltési oldalán választanunk kell a Unix, Linux és általános forráskód közül, és meg kell szereznünk ***Ryouichi Suzuki Density File*** kiterjesztés-foltját (lásd a hálózaton elérhető címek között).

Ez egy ***Zip*** fájl, amely a POV-Ray fájljainak egy részét helyettesítő forráskódot tartalmaz. A ***pov35dfjs.zip*** fájlt a ***povray-3.50c/src*** könyvtárban kell kicsomagolni, ahol felül fog írni tizenhárom fájlt.

A jelenetek és a szintfelületek

A ***POV-Ray*** olyan jelenetleíró fájlokkal dolgozik, amelyek minden információt tartalmaznak a bittérképes kép létrehozásához. A ***POV-Ray*** saját, a honlapján jól dokumentált jelenetleíró nyelvet használ. Ha korábban nem használtunk semmilyen sugárkövető programot, azt javaslom, hogy ismerkedjünk meg a módszer alapjaival és a ***POV-Ray*** jelenetleíró fájljaival, még mielőtt a forráskódhoz hozzányúlnánk. A ***POV-Ray***-ben leképezett elemeket objektumoknak nevezzük, amelyekre példa a téglatest, gömb, tórusz vagy a sík. A felhasználó megadja az objektum jelenetben elfoglalt helyét, a méreteit, színeit, megvilágítási jellemzőkét és így tovább. Ezek az adatok egy jelenetleíró fájlban szerepelnek, amelyeket a ***.pov*** kiterjesztésük miatt esetenként ***pov***-fájloknak is neveznek. A jelenetleíró fájlok közönséges szövegfájlok, amelyek értelmezését a ***POV-Ray*** futásidőben végzi el.



1. ábra Egy teljes szupercellás vihar légi felvétele körülbelül 30 kilométeres távolságból a POV-Ray segítségével leképezve

Egy általánosan használható objektum a szintfelület. Ez egy olyan háromdimenziós forma, amelynek a felülete az azonos függvényértékkel rendelkező pontok megjelenítésével áll elő. A függvény állandó értékét a felhasználó választhatja ki. A ***POV-Ray*** számos olyan előre definiált objektumot tartalmaz, amely valójában szintfelületnek is tekinthető. A következő jelenetleíró fájlból származó részlet például egy 0,7 egység sugarú, szürke színű gömböt képez le, melynek középpontja az origóban, vagyis a `(0,0,0)` Descartes-koordinátákkal jellemzett pontban helyezkedik el:

```
sphere
{
    <0,0,0>, 0.7
    pigment { rgb .5}
}
```

Ugyanez az objektum szintfelületként is megadható lenne az alábbi módon:

```
#declare R = 0.7
isosurface
{
    function { x*x + y*y + z*z - R*R}
    pigment { rgb .5}
}
```

A meghatározás hátterében az áll, hogy az R sugarú gömb az alábbi matematikai egyenlettel írható le:

$$x^2 + y^2 + z^2 - R^2 = 0$$

A szintfelületeknek ez a sokoldalúsága volt az a tulajdonság, ami miatt ezt az objektumot választottam a zivatarok ábrázolásához.

Sűrűségfájlok

A gömb példájában egy matematikai függvényt használtam a szintfelület értékének kiszámításához. A zivatarjaimhoz használt numerikus modell adatai nem írhatók fel egyetlen matematikai függvényként, ehelyett egy olyan háromdimenziós lebegőpontos tömbökről van szó, amelyek minden rácspontban valamilyen modellváltozót tartalmaznak, ami például lehet hőmérséklet, szélsebesség vagy felhősűrűség (*2. ábra*).

A ***POV-Ray 3.5*** változatának van egy sűrűségfájl nevű szolgáltatása, amely lehetővé teszi a függvények rácspont-értékekként való leképezését. A ***POV-Ray*** dokumentációja a következőképpen írja le a sűrűségfájlokat: „A sűrűségfájl-mintázat egy olyan háromdimenziós bittérkép-mintázat, amely egy ***<0,0,0>*** és ***<1,1,1>*** koordinátájú pontok között elhelyezkedő egységnyi kockát tölt be. Az adatfájl egy a ***POV-Ray*** számára készült, ***df3*** nevű nyers bináris fájlformátum."

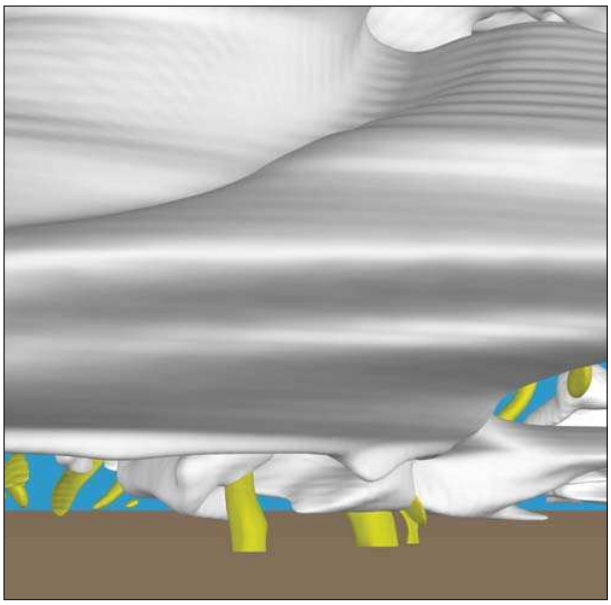
A sűrűségfájlok olyan függvényként használhatóak, amelyeket a szintfelület-objektum paraméterként kap meg. Íme egy példa a szintfelület leképezéséhez használt sűrűségfájlra:

```
#declare DENSFUNC=function
{
    pattern
    {
        density_file df3 "cloud.df3"
        interpolate 1
    }
}
isosurface { function {  0.1 - DENSFUNC(x,y,z) }
```

A fenti példában a ***cloud.df3*** fájl segítségével egy 0,1 értékű szintfelület állítanánk elő egy háromvonalas (trilineáris) interpolációs séma használatával (az interpolációról rövidesen lesz még szó).

A sűrűségfájl formátuma szigorúan kötött, az adatokat 8 bites értékek képviselik (0 és 255 közti előjel nélküli egészek), amelyeket a program alakít át 0,0 és 1,0 közti értékekre. Mivel az én zivatar-adataim 32 bites lebegőpontos értékek, a sűrűségfájl-formátum nem alkalmas közvetlenül az eredeti ***POV-Ray*** 3.5 változattal való használatra.

Itt lép be ***Ryouichi Suzuki***, aki 1996 óta fejleszt a ***POV-Ray*** számára kiegészítő kódokat. Ő készítette a ***POV-Ray*** 3.0-hoz azokat a foltokat is, amelyek elsőként vezették be a 3.5 változatba már beépített objektumként szereplő szintfelület-objektumokat. ***Suzuki*** fent említett ***zip***-fájljában lévő kód tartalmaz olyan eljárásokat is, amelyek kiterjesztik a ***POV-***



2. ábra Egy példa többszörös szintfelületekre, ahol a központban a szupercella felhőfalnak nevezett tartománya áll. A felhőfal alatt az előtérben látható sárga szintfelületek a tornádószerű örvénylő mozgást mutatják.

Ray sűrűségfájljainak lehetőségeit, képessé téve a programot többek közt a lebegőpontos sűrűségfájlok alapján történő leképezésre is.

Amikor sűrűségfájlokat használunk függvények helyett, felmerülhet valakiben a gondolat, hogy míg a függvények folytonos kifejezések – vagyis az ***x***, ***y*** és ***z*** térkoordinátákhoz bármilyen lebegőpontos érték tartozhat – a sűrűségfájl az adatoknak olyan diszkrét halmaza, amelyet a tömbök egész indexei jellemeznek. Egy kép leképezésénél a rácspontok közti területet interpoláció segítségével kell kitölteni. A két rendelkezésre álló interpolációs eljárás a háromvonalas interpoláció és a háromirányú köbös simulógörbe (tricubic spline). A háromvonalas interpoláció gyorsabb, de gyakran nem ad olyan egyenletes eredményt, mint a simulógörbe.

A modelladatok betöltése a POV-Ray-be

Miután a ***POV-Ray 3.5***-re feltelepítettük ***Suzuki*** sűrűségfájlra vonatkozó foltjait, a rendszer készen áll arra, hogy lebegőpontos értékek alapján képezzen le szintfelületeket – amennyiben az adatok ***df3*** formátumban, vagy ***Suzuki*** kiterjesztett formátumában rendelkezésre állnak. Az én esetemben az adatok több száz gigabájtnyi *HDF*-fájlként (hierarchical data format, hierarchikus adatformátum) tárolódnak, amely formátum kifejezetten numerikus modellek adatainak a tárolására lett kifejlesztve. Mivel engem nem csak néhány szintfelületes kép előállítása érdekelt, hanem szerettem volna az előállított több száz, esetleg több ezer képből animációt is létrehozni, a ***HDF-df3*** átalakítás nem jöhetett szóba. Ehelyett inkább vizsgálni kezdtem a ***POV-Ray*** eljárásait, amelyekkel a sűrűségfájlokat kezeli és reménykedtem, hogy sikerül a kódot úgy módosítanom, hogy olvasni tudja a *HDF*-fájljaimat.

Fontos volt számomra az is, hogy a módosításaim ne csökkentsék az eredeti program használhatóságát és teljesen kompatibilis maradjon a folt nélküli változattal. A célomat úgy értem el, hogy a jelenetleíró fájlomhoz olyan új objektumokat adtam, amelyeket a módosított változat képes volt elemezni és leképezni, míg az összes többi objektum változatlan maradt. Az általam módosított kód központi része a ***pattern.cpp*** fájlban található, amely a `Read_Density_File` eljárást tartalmazza. Ahogy a neve is sejteti, ez az eljárás olvassa be a sűrűségfájlt egy háromdimenziós tömbbe. Ennek az eljárásnak a mintájára létrehoztam egy új eljárást `Read_Hdf_File` néven, amely az én leírófájlomat olvassa be a ***POV-Ray***-be. Ez az a rész, ahol a legtöbb módosítást igényli a programkód, ha ragaszkodunk a saját adatformátumunkhoz. Az *1. listán* a `Read_Hdf_File` eljárás rövidített változatát láthatjuk.

A `Read_Hdf_File` függvény olvassa be a *HDF* lebegőpontos adatait a `mapF` nevű háromdimenziós lebegőpontos tömbbe, amely ezután már sűrűségfájlként kezelhető. A ***hystory.c*** fájlban külön írtam meg azokat a kódokat, amelyekre a *HDF* I/O-eljárásai hivatkoznak a ***pattern.cpp*** fájlban. A saját adatfájl-formátumunkhoz saját magunknak kell megírnunk azt a formátumra jellemző kódot, amely a háromdimenziós adatainkat beolvassa a ***POV-Ray***-be.

Módosítanom kellett még néhány fájlt annak érdekében, hogy a ***POV-Ray*** felismerje a *HDF* fájlformátumot és hogy lehetővé váljon képenként egynél több modellváltozó leképezése. Az *1. táblázat* sorolja fel a módosított fájlokat és rövid leírást ad az elvégzett változtatásokról.

A *HDF* fájl a sűrűségfájllal ellentétben lehetővé tesz fájlonként egynél több változó tárolását is. Az esetemben minden egyes *HDF* fájl a modell egy bizonyos időpontbeli állapotát írja le, és fájlonként 12 háromdimenziós változót tartalmaz. Gyakran sokkal látványosabb, ha egy képen több változót (felhő, eső, jégeső, hó) együtt is megfigyelhetünk. Ennek megvalósítására a *HDF* fájlformátum ábrázolásához egy új tokent, a ***HDF_TOKEN***-t hoztam létre (szemben az eredeti df3 ábrázolását végző DF3_TOKEN nevű tokennel), és egy Var nevű új karaktertömböt hoztam létre a `Density_file_Data_Struct` szerkezetben. A Var a jelenetleíró fájlban kap értéket, és a *HDF*-eljárásoknak paraméterként átadva meghatározza, hogy melyik modellváltozó legyen kiválasztva. A karakterláncként tárolt változónév értelmezéséhez egy további `case` utasítást adtam a ***parstxtr.cpp*** fájl `Parse_PatternFunction` függvényéhez (*2. lista*). Láthatjuk a `Parse_Comma` és `Parse_C_String` hozzáadását is, amelyek a beolvasandó változót csípik el.

A jelenetleíró fájl

Most már minden részlet a helyén van ahhoz, hogy létrehozzunk egy ***POV-Ray*** által értelmezhető jelenetleíró fájlt. Ehhez a ***Suzuki***-féle sűrűségfájl-kiterjesztés honlapján megtalálható példafájlt használtam sablonként egy kicsit az igényeimhez igazítva. A *3. lista* tartalmazza a teljes jelenetleíró fájlt, amit az 1. ábrán látható kék hátterű és burkolófelülettel ellátott felhősűrűség-szintfelület leképezéséhez használtam. A tetejétől indulva először a #version utasítást látjuk, ez teszi lehetővé a nem hivatalos ***POV-Ray*** változatom működését. Az ezután következő kilenc `#declare` utasítás a szintfelületet tartalmazó téglatestet és a tengelyek beosztását határozza meg.

1. lista A Read_Hdf_File listájának rövidített változata a pattern.cpp fájlból

```
void Read_Hdf_File (DENSITY_FILE * df)
{
 Locate_Density_File(df->Data->Name);
 df->Data->Type = 1; //floating point data
 Open_HDF_File(df->Data->Name);
 //povray needs array dimensions
 Read_HDF_File_Geometry(nx,ny,nz);
 df->Data->Sx = nx;
 df->Data->Sy = ny;
 df->Data->Sz = nz;
 //this array will contain density file data
 Allocate_Memory(mapF,nx,ny,nz);
 //read variable into mapF array
 Get_HDF_File_Data(Var,mapF,nx,ny,nz);
 //density file pointer now points to model data
 df->Data->DensityF = mapF;
}
```

2. lista A HDF_TOKEN case-ágában szükség van egy további elemző részre, amely lehetővé teszi annak megadását, hogy melyik változó leképezése történjen. A kódrészlet a parstxtr.cpp fájl Parse_PatternFunction eljárásában található.

```
EXPECT
  CASE (DF3_TOKEN)
    New->Vals.Density_File->Data->Name =
        Parse_C_String(true);
    Read_Density_File(New->Vals.Density_File);
    EXIT
  END_CASE
  CASE (HDF_TOKEN)
    New->Vals.Density_File->Data->Name =
        Parse_C_String(true);
    Parse_Comma();
    New->Vals.Density_File->Data->Var =
        Parse_C_String(true);
    Read_Hdf_File(New->Vals.Density_File);
    EXIT
  END_CASE
```

Továbbhaladva a jelenetleíró fájlban a színek és a felület kidolgozásának paramétereinek, valamint a kameraállás és megvilágítás jellemzőinek megadása történik. Az ezt követő sorok tartalmazzák a lényeget, a szintfelület meghatározását. A `QCFUNC` egy függvény, amely a forrásadatokat a ***supercell.ck10990.hdf*** nevű ***HDF***-fájlból olvassa be és ezekből a leképezéshez a `QC` változó tartalmát használja fel (amely változó a felhősűrűséget tárolja). Az interpolációhoz a köbös simulógörbét választjuk, és a teljes tartományra érvényesítjük a fokbeosztást, így az összes olyan térbeli koordináta, mint a kameraállás, megvilágítás helyzete, egybeesik az adatértékekkel. Alapértelmezésben a ***POV-Ray*** tartománya mindhárom irányban a 0,0-tól 1,0-ig terjedő tartományt használná.

1. táblázat ***A modelladatok POV-Ray-ben történő alkalmazását biztosító módosítások listája***

Fájl	Módosítás
pattern.cp	A modelladatokat beolvasó Get_HDF_File_Data eljárás hozzáadása.
pattern.h	A Read_Hdf_File meghatározásának hozzáadása.
parstxtr.cpp	A HDF_TOKEN case-blokkjának hozzáadása.
tokenize.cpp	A HDF_TOKEN hozzáadása a Reserved_Words tömbhöz.
frame.h	A char *Var1 hozzáadása a Density_file_Data_Struct szerkezethez.
parse.h	A HDF_TOKEN hozzáadása a TOKEN_IDS-hez..

Létrehoztam egy `QCISOSFC` nevű makrót, amely paraméterként a leképezni kívánt szintfelület értékét és a szintfelület átlátszóságát kapja meg. Az szintfelület átlátszósága egy jól használható tulajdonság, amikor két egymást fedő szintfelületet ábrázolunk. Például érdemes áttetszővé tenni a felhőt, amikor egyidejűleg jégsűrűség-szintfelületet is ábrázolunk, mivel a jég gyakran található a viharfelhők belsejében.
A makróban leképezés szintfelület-függvényeként a feljebb definiált `QCFUNC`-ot választjuk. A kiválasztott szintfelület azokat a felhősűrűség-értékeket veszi figyelembe, amelyek nagyobbak a felülethez kiválasztott 0,0002 értéknél.
A `max_gradient` paraméter lényegében azt határozza meg a ***POV-Ray*** számára, hogy mennyi munkát fordítson a szintfelület meghatározására. A működés szempontjából azt rögzíti a program számára, hogy mi az a maximális gradiens (változási gyorsaság a távolság függvényében), amellyel a függvény egy adott szintfelület-pont környezetében lévő felület-adatokat leírja. Ezt a számot nagyon körültekintően kell megválasztani. Túl kicsire választva az értéket lyukak lesznek a felületben, esetleg egyáltalán nem kapunk felületet; túl nagyra választva viszont a ***POV-Ray*** sokkal hosszabb ideig fog futni, mint az szükséges lenne. Némi gyakorlat kell ahhoz, hogy a megfelelő értéket válasszuk. Én a 0,0002 értéket választottam, ami a felhősűrűség körülbelül 0,0-tól 0,01 értéktartományához viszonyítva első ránézésre kicsinek tűnik. A ***POV-Ray*** egy kép túl nagy vagy túl kicsi értékkel végrehajtott leképezése után javasol egy olyan értéket, amit a leképezés alapján megfelelőnek tart.

Képek és egyebek létrehozása

A program változtatásokkal történő lefordításához néhány kisebb módosítást kell végrehajtanunk az ***src/Makefile*** fájlban, amely a ***POV-Ray*** könyvtárának felső szintjén lévő `configure` első futtatásakor jön létre. Ez az eset akkor áll fenn, ha az adatfájljaink beolvasó eljárásaihoz külső programkönyvtárakat használunk, vagy ha a fájlok I/O műveleteit külön kóddal oldottuk meg.

3. lista A cloud.pov fájl

```
#version unofficial dfe 3.5;
#include "colors.inc"

#declare x0 = 0.0;
#declare x1 = 700.0;
#declare y0 = 0.0;
#declare y1 = 600.0;
#declare z0 = 0.0;
#declare z1 = 80;

#declare scalex = (x1-x0+1);
#declare scaley = (y1-y0+1);
#declare scalez = (z1-z0+1);

#declare R = 0.7;
#declare G = 0.7;
#declare B = 0.7;

#declare AMBIENT    = 0.5;
#declare DIFFUSE    = 1.1;
#declare SPECULAR   = 0.3;
#declare ROUGHNESS  = 0.01;
#declare BRILLIANCE = 1.0;

camera {
    up           <0,0,1>
    sky          <0,0,1>
    right        <3.0,0,0>
    direction    <1.0,0,0>
    location     <420,70,70>
    look_at      <370,300,90>
}

light_source { <100,100,100> color Gray25
shadowless}
light_source { <400,200,30> color Gray20 }
light_source { <1000,-500,150> color Gray25 }
light_source { <-400,-500,150> color Gray25 }

#declare QCFUNC = function {  pattern{
    density_file hdf "supercell.ck10990.hdf","QC"
    interpolate 2 //tricubic spline
    frequency 0
scale <scalex,scaley,scalez> }  }

#macro QCISOSFC(iso,trans)
isosurface{  function{  -QCFUNC(x,y,z) }
threshold -iso
max_gradient 0.0002
contained_by{ box{ <x0,y0,z0>,<x1,y1,z1>} }
texture{  pigment{ color rgbt<R,G,B,trans>}
finish{ ambient AMBIENT diffuse DIFFUSE
    specular SPECULAR roughness ROUGHNESS
    brilliance BRILLIANCE}   } _shadow
}
#end

QCISOSFC(0.0002,0.0) // render cloud

box {  <x0,y0,z0> <x1,y1,z0> // tiles 5km square
pigment { checker color NewTan,
    color .90*NewTan scale 50}
finish { ambient 0.5 diffuse 0.5}  }

background { SkyBlue}  // what else?
```

A program lefordítása után a parancssorból indíthatjuk a *POV-Ray*-t. Az alábbi parancs beolvassa a *cloud.pov* fájlt és egy *600x400* felbontású élsimított *PPM*-fájlt állít elő megjelenítve a képernyőn a leképezés folyamatát:

```
/home/orf/povray-3.50c-orf/src/povray +D \
Input_File_Name=cloud Width=600 Height=400 \
Antialias=on Output_File_Type=P
```

Miután az adataink alapján a *POV-Ray* sikeresen elkészítette a képet, a beállítható eszközök széles választékával alakíthatjuk a leképezést az igényeinknek legmegfelelőbb formára. Ha folyamatosan változó adatokkal rendelkezünk, kézenfekvő igény, hogy ezekből mozgóképet hozzunk létre. Én *Python* parancsfájlokkal oldottam meg a különböző *POV-Ray* példányok indítását a leképezőtelepem egyes processzorain, ahol a processzorok párhuzamosan dolgoznak a modell különböző időállapotainak adatain. A kapott *PPM*-fájlokat ezután az `mjpegtools` segítségével illesztem össze mozgóképpé. A kutatói honlapomról letölthető néhány ilyen animáció. Sztereóképeket és mozgóképeket is létrehoztam az osztályunk *GeoWall* rendszere számára. A sztereó képpárok létrehozása a *POV-Ray* számára nem jelent gondot és valóban új megvilágításba helyezi az adatainkat. A *POV-Ray* használata a modelljeim adatainak megjelenítésére egy sereg új izgalmas lehetőség felé nyitotta meg számomra az utat, s remélem, hogy ebben nem vagyok egyedül.

A cikkhez kapcsolódó hivatkozások a ➲ *www.linuxjournal.com/article/7754* címen érhetőek el.

Linux Journal 2004. november, 127. szám

Leigh Orf a Michigani Központi Egyetem légkörkutatással foglalkozó professzora, és hosszú ideje Linux felhasználó. Tudományos kutatási területei közé tartozik a viharok valósághű szimulációja és megjelenítése, amelyekhez nagy teljesítményű Linux-fürtöket használ. Szabadidejében élvezettel főzi saját sörét, rádióamatőrködik, szaxofonozik vagy indul sátortúrára feleségével, Annie-vel.
A leigh.orf@cmich.edu címen érhető el.

PHP SQLite adatbázisháttérrel

Eddigi PHP alapokra épülő munkáim során ritka volt az olyan feladat, ahová ne jött volna jól egy kis adatbázis támogatás a hatékony adatkezelés érdekében. Ámde a PHP-ben legyártott alkalmazások nagy része olyan környezetben éli életét, ahol az adatbázis kiszolgáló ugyanazon a gépen található, vagy esetleg egy ugyanazon hálózaton figyelő másikon.

Bevezetés

Bár az adatbázis szerverek maguk lehetővé teszik, az ilyen kis és közepes projektek nem igényelnek többet egy darab adatbázis hozzáférést biztosító felhasználónév/jelszó párosnál. De mivel a nyers fájlokkal való izommunka mégsem olyan csábító választási lehetőség, marad az *SQL* adatbázis szerverek dolgoztatása, felesleges hálózati rétegekkel, széleskörű jogosultság rendszerekkel. Ez így is ment mindaddig, míg csak egy napon elém nem került az ***SQLite***, a beágyazható adatbáziskezelő motor. Bár nem lehet belőle nagyvállalati központi adattár kiszolgálót építeni, de saját kis webhelyeink blogjainak, fórumainak, on-line boltocskáinak meghajtására kiválóan alkalmas.
Ez a cikk azok számára lehet hasznos olvasmány, akik már foglalkoztak a *PHP* valamely adatbáziskezelő kiterjesztésével, mivel javarészt inkább csak a különbségek, egyedi megoldások bemutatására szorítkozom.

Telepítés, rendszerigény

A *PHP SQLite* kiterjesztésének birtokbavétele nem tartozik a keményebb rendszergazdai feladatok közé. Az *SQLite* tervezésekor az egyik fő szempont pont az volt, hogy könnyen illeszthető legyen bármihez, és lehetőleg ne függjön léte más függvénykönyvtáraktól. Így ahol a *PHP* futásképes, ott az *SQLite*-al sem lesz gond. Munkába fogásához csak a megfelelő *PHP* modul telepítése szükséges. A legtöbb kiterjesztéssel ellentétben ennek telepíthetősége nem függ külső függvénykönyvtár meglététől, mivel a *PHP* kiterjesztés magában foglalja a teljes *SQLite* függvénykönyvtárt. Nos, nem hiába hívják beágyazható adatbáziskezelőnek.
A *PHP 5* esetében az alaptelepítéssel már kezünkben is van a teljes *SQLite* eszköztár telepítve, élesítve. A 4-es változatok esetében a *PHP Pear* névre hallgató csomagkezelőjét kell kicsit munkára fognunk, a

```
pearinstall sqlite
```

utasítás parancssorba gépelésével, minek eredményeként letöltésre, majd gépünkön forrásból fordításra kerül a megfelelő PHP kiterjesztésmodul. A sikeres fordításhoz a gcc mellett szükség lesz az autotools (autoconf, automake) eszközökre is.
Korábbi *PHP 4*-es változatok esetén előfordulhat, hogy egy kellemetlenkedő,

```
No releases found for package 'sqlite'
```

üzenetet kapunk próbálkozásunkra. Semmi gond, csak a *PHP*-vel kapott *Pear* telepítő kicsit öreg, így előbb saját maga frissítésére kell rávennünk. A következő parancs hatására ez meg is történik, ami után az ***sqlite*** csomagot is meg fogja találni:

```
pear upgrade Console_Getopt PEAR Archive_Tar
```

Siker esetén helyére kerül az ***sqlite.so*** állomány, melyet vagy a ***php.ini*** egy megfelelő `extension=sqlite.so` sorával, vagy a *PHP* parancsállományaink elejére tett `dl('sqlite.so')` utasítással tudjuk a PHP részévé tenni. Innentől kezdve rendelkezésünkre áll az ***sqlite*** parancskészlet.

Gyorstalpaló

Az *SQLite* – hasonlóan a *PHP*-hez – nem kifejezetten típusérzékeny. Valójában jórészt magasról tesz rá, milyen típust adtunk meg melyik oszlopnál, egy `INT` típusú mezőbe büntetlenül tehetünk szöveget is. A szövegesnek megjelölt oszlopok méretmeghatározása is inkább csak arra jó, hogy magunknak dokumentáljuk, körülbelül milyen adatokat is szeretnénk tárolni. Hasznos információvá szinte csak a tábla neve és az oszlopnevek válnak. Bizonyos esetekben, például rendezések során persze szerepet kap a megadott oszloptípus, ami szerint számszerűleg, vagy szövegként kezelve rendezi sorba az elemeket.
Aki már foglalkozott a *PHP* és valamely adatbázis kiszolgálótípus házasításával, az *SQLite* modul parancskészlete sem fog nagy meglepődéseket okozni. Sajnos ez a modul is a maga útját járja, így egyik jelenlegi *SQL* motorral dolgozó kiterjesztésnek sem felel meg egy az egyben parancskészlete.

1. lista

```
<?php
if (!function_exists('sqlite_fetch_single')) {
    function sqlite_fetch_single($res) {
        $row = sqlite_fetch_array($res,
            ➥SQLITE_NUM);
        return $row[0];
    }
}
?>
```

Mivel nincs külön adatbázis szerver, kapcsolódnunk nem kell hozzá, valamint jelszó sem kell az adatbázis eléréséhez. Ez a művelet inkább hasonlatos ahhoz, ahogy egy fájlt megnyitunk:

```
$db = sqlite_open(<fájlnév>, [fájljogok],
➥[&hibaüzenet]);
```

Az egyetlen, amit mindenképp meg kell adnunk, az ***SQLite*** adatbázist megtestesítő fájl neve, esetlegesen az elérési útjával egyetemben. A második paraméter alapértéke a 0666 oktális érték, ami a `rw-rw-rw-` fájljogosultságnak felel meg. Jelenleg ezen paraméternek nincs jelentősége, bármit is adunk meg, az ***SQLite*** ezzel a móddal fog próbálkozni. Harmadik paraméterként megadhatunk egy változót, amibe az esetleges szöveges hibaüzenet kerül, ha az adatbázis megnyitása kudarcba fulladt volna. Ilyen esetekben az `sqlite_open()` amúgy `false` értékkel fog visszatérni. Amennyiben a megadott helyen nem található a keresett fájl, ezen utasítással létre is hozzuk azt, feltéve ha erre a könyvtárra a webszerver felhasználója megfelelő jogosultságot kap. Fontos tudni, hogy nem elég, ha a létrehozás idejére adunk csak írási jogot a hordozó könyvtárra. Az adatbázis módosításához, bővítéséhez ugyanis nem elég, ha magát a fájlt tudjuk írni, mivel ezen műveletek során ugyanitt ideiglenes állományok létrehozására is szüksége lesz ***SQLite*** motorunknak. Amint sikeresen rendelkezünk egy megnyitott adatbázissal, minden úgy megy, ahogy azt más adatbázisoknál is megszokhattuk. Az SQL kérések futtatására az `sqlite_query()` valamint az `sqlite_exec()` szolgál, míg az eredménylisták kisajtolására az `sqlite_fetch_*` függvények. Munkánk végeztével egy esetleges `sqlite_close()` által felszabadíthatjuk lefoglalt erőforrásainkat.

Az `sqlite_query()` és `sqlite_exec()` is ***SQL*** parancsok végrehajtására szolgál, de csak az első ad vissza eredménylistát. Az `sqlite_exec()` csupán igaz/hamis értéket ad attól függően, hogy a parancs futtatása sikeres volt, vagy sem. Ezen függvények két adatot várnak, a megnyitott adatbázisra mutató erőforrás azonosítót, valamint magát az ***SQL*** kérést. A két paraméter tetszőleges sorrendben megadható, a kézikönyv az adatbázis azonosító előre vételét javasolja. Aki sokat dolgozott például ***mysql***-el, esetleg kényelmesebbnek találhatja az ott megszokott sorrendet. Egy paranccsal egyszerre több ***SQL*** kérést is futtathatunk, ezeket az `sqlite_exec()` minden esetben futtatni fogja maradéktalanul. Az `sqlite_query()` csak akkor, ha a visszakapott eredményazonosítót a későbbiek folyamán nem használjuk fel. Ha mégis, akkor csak az első kérés fog végrehajtódni. Sikeres lekérdezés után jön az adatkinyerés az eredménylistából. Teljes, több mezős adatsort, az `sqlite_fetch_array()` függvénnyel tudunk olvasni. Első és egyetlen kötelező paraméterként az eredménylista azonosítót várja. Ellentétben más adatbázis kiterjesztésekkel, itt nem áll rendelkezésre `sqlite_fetch_row()` függvény. Erre is az előbbi alkalmazandó, a második paraméterben megadva, milyen formában kérjük az adatsort. Ehhez az ***SQLite*** kiterjesztés három előre megadott állandót alkalmaz:

2. lista

```
<?php

/* Ideiglenes adatbázis létrehozása tesztadatokkal */
$db = sqlite_open(':memory:');
sqlite_exec($db, 'CREATE TABLE ertekek (szam)');
foreach (array(14, 33, 62, 732, 1, 57, 21) as
        ➥$szam) {
    sqlite_exec($db, 'INSERT INTO ertekek VALUES
            ➥(' . $szam .')');
}

/* Saját (lentebb megvalósított) függvényeink
átadása SQLite-nak */
sqlite_create_aggregate($db, 'avg',
➥'avg_feldolgozo', 'avg_osszegzo');
sqlite_create_function($db, 'dup', 'dup');

/* Lekérdezés, melyben használjuk mindkét függvényt */
$res = sqlite_query($db, 'SELECT avg(dup(szam))
                ➥FROM ertekek');
echo sqlite_fetch_single($res);
sqlite_close($db);

/* Az SQLite számára elállított függvények */
function dup($ertek) {
    return $ertek * 2;
}

function avg_feldolgozo($kornyezet,
                    ➥$aktualis_ertek) {
    $kornyezet['osszeg'] += $aktualis_ertek;
    $kornyezet['elemek'] += 1;
}

function avg_osszegzo($kornyezet) {
    return ($kornyezet['osszeg'] /
            ➥$kornyezet['elemek']);
}

?>
```

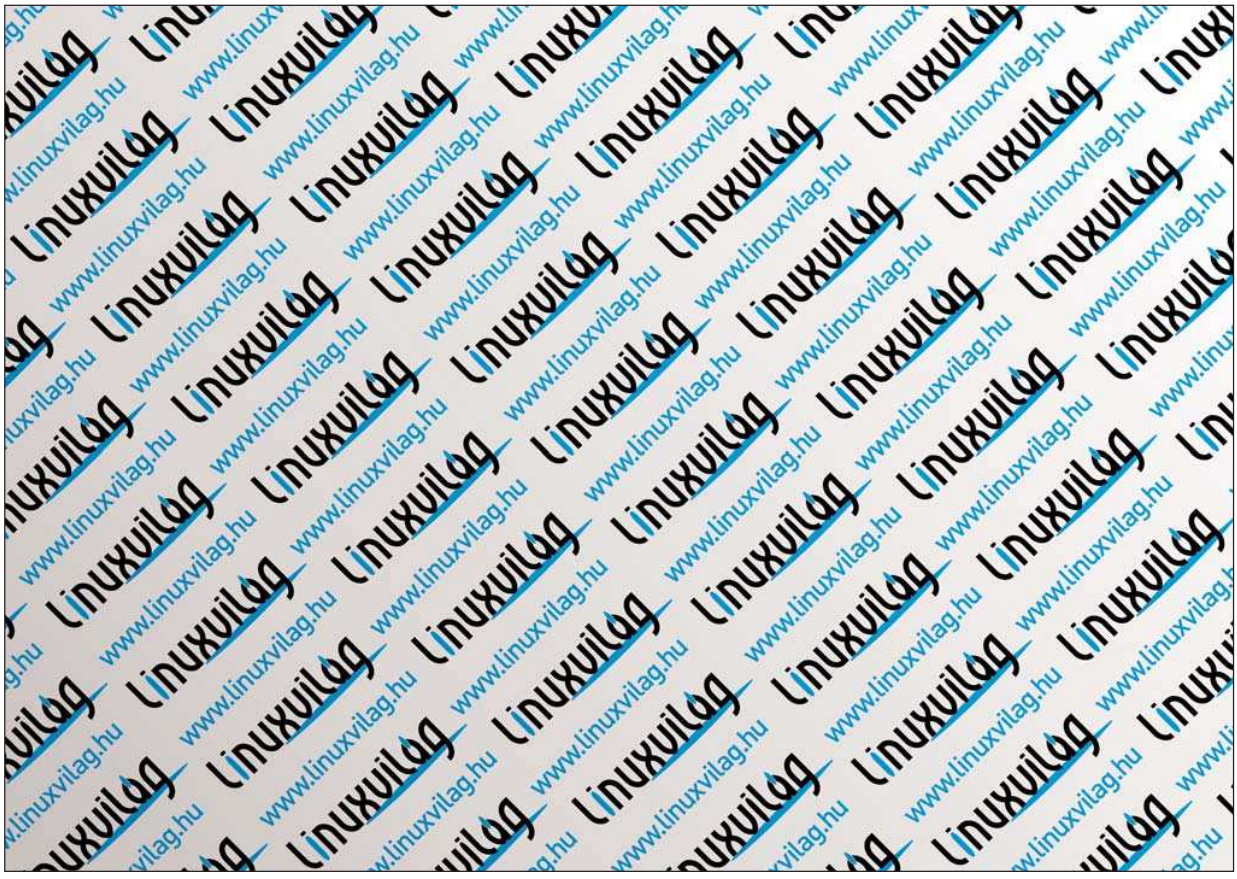
- `SQLITE_ASSOC` – a mezőnévnek megfelelő indexre teszi az értékeket
- `SQLITE_NUM` – a mező sorszámának megfelelő indexre teszi az értékeket
- `SQLITE_BOTH` – mind név, mind oszlopszám szerint is elhelyezi az adatokat

Ha amúgy is egy nagy tömbbe szeretnénk ciklikusan beolvasni a teljes eredménylistát, érdemes `sqlite_fetch_all()`-ra bízni ezt. Ez a teljes eredménylistát beolvassa egy tömbbe, melynek minden eleme egy adott sor lesz. Paraméterezésre teljesen megegyezik az előzővel. Mivel ekkor a teljes visszakapott adatsor a memóriába kerül, nagy listák lekérésekor ez a módszer nem javallott. Általában azonban igaz, hogy 50 sornyi adatnál többet amúgy sem illik a látogató orra alá dörgölni egyszerre.
Az ilyen tömbbe lekérdezésre van viszont az ***SQLite*** kiterjesztésnek egy még kényelmesebb változata. Ezt `sqlite_array_query()` néven érhetjük el és egyesíti az `sqlite_query()` valamint az `sqlite_fetch_all()` függvényeket. Első két paramétere a kérésfuttató függvényekével egyezik meg, harmadikként pedig megmondhatjuk, milyen formában kérjük az adatokat a tömbbe helyezni.
Számomra szimpatikus, hogy az egyetlen adat lekérését megkönnyítendő, létezik egy `sqlite_fetch_single()` függvény. Ez sajnos a ***PHP 4***-es változatokhoz telepíthető kiterjesztésben nincs benne, de a ***PHP 5*** tartalmazza.
Az *1. lista* bemutat egy egyszerű kódocskát mellyel eme hiány 4-es változatokban áthidalható.

Ennyivel a mindennapokban nagyjából el lehet boldogulni, csupán csak a hibakeresésről nem esett szó. Ez itt két lépésben történik, először meg kell tudnunk a feltételezett hiba kódját. Erre az `sqlite_last_error()` függvénnyel nyílik lehetőség. Egyetlen paraméterében sajnos mindenképp meg kell adni, mely adatbázissal kapcsolatban érdekel ez az adat. Máshol, például a ***MySQL*** kiterjesztés esetében alapértelmezésként a legutóbb megnyitott adatbázist veszi elő ilyen adat hiányában. Persze egy hibát azonosító szám nem sokat segít a hiba felderítésében. Többre jutunk, ha a hibakódot az `sqlite_error_string()` függvénnyel lefordíttatjuk emberi nyelvre. A hibakódért cserébe ez annak szöveges változatát adja vissza.

Szorosabb együttműködés

A kisebb erőforrás igények mellett a beágyazottságnak van még egy nagy előnye. ***SQLite*** kéréseinkben tetszőleges bonyolultságú, ***PHP***-ben írt függvényeinket is alkalmazhatjuk. Néhány alapvető függvénnyel az ***SQLite*** is rendelkezik, de akár ezek felülírására is képesek vagyunk. Ehhez először is létre kell hoznunk egy, a célt megvalósító ***PHP*** függvényt, mely a bejövő adat(ok)on tetszőleges műveleteket végezve végül visszaad egy értéket. Ez csak az ***SQLite*** által is értelmezhető adat lehet, azaz nem adhat vissza tömböt, objektumot, erőforrás hivatkozást. Ha függvényünk harcra kész, az `sqlite_create_function()` függvény segítségével tudathatjuk, hogy azt az ***SQLite*** lekéréseiben alkalmazni szeretnénk. Első paramétereként az adatbázis azonosítót, másodikként és harmadikként pedig egy-egy függvény nevet.



Először a függvény lekérésekben használatos nevét, majd a már létrehozott *PHP* megvalósítás nevét kell megadni. Negyedik adatként megadhatjuk az *SQLite* motornak, hány paramétert várjon el ez a függvény.
Az már csak a fantázián múlik, a függvényen belül mihez kezdünk a kapott adattal. Akár más típusú adatbázisokhoz is hozzáférhetünk, IP cím alapján kideríthetjük a hozzá tartozó gépnevet. Persze a lekérés hatékonysága a feladatok bonyolultságának növekedtével alaposan visszaeshet.
Nem csupán az egyes értékek módosítására van lehetőségünk, de a `COUNT()`, `SUM()` és hasonló, a teljes eredménylistát átfogó függvényeket is létrehozhatunk *PHP* nyelven. Az `sqlite_create_aggregate()` által hozhatjuk ezt adatbázisunk tudtára. Elsőként ez is az adatbázist kéri beazonosítani, másodikként pedig a majdani *SQL* kérésekben alkalmazott függvénynevet kell meghatározni. Az átfogó függvény megvalósításához azonban két különálló függvényre lesz szükség. Ezeket adhatjuk meg a harmadik, illetve a negyedik paraméterekben. Így ötödik helyre csúszik annak megadási lehetősége hány paramétert is vár függvényünk.
A megadandó függvénypár első tagja minden egyes kapott soron végrehajtásra kerül, majd végül az adatgyűjtés végeztével a második függvény fogja megmondani a választ. Első paraméterként mindkét függvény egy tetszőlegesen felhasználható, referenciaként kezelt változót kap, nevezzük ezt „környezetnek”. Ebben a sorok feldolgozása idején tetszőleges méretű adathalmazt tárolhatunk. Végül az összegző függvényebből okoskodhatja ki az eredményt. Ha lehetséges, érdemes amit csak lehet már menet közben kiszámítani és minél kevesebb nyers adatot tárolni feleslegesen. Ugyanis nagy lekéréseknél egy hatalmasra növekvő tömb csak kiverné a biztosítékot. Az összegző függvénynek nincs is szüksége más adatra, mint erre a környezetleíró adat(halmaz)ra. A soronként meghívott függvény ezután minimum egy, de akár több adatot is várhat, szükség szerint. A 2. *lista* egy nem túl bonyolult példát mutat mind az egyedi adatokkal dolgozó, mind az átfogó függvények létrehozására és alkalmazására.

Tippek, tanácsok

Az egyik legszembetűnőbb hiányosság a jelenlegi *SQLite* motornak az `ALTER TABLE` megvalósításának teljes hiánya. Meglévő tábláink felépítésének módosítása így nem egy egy sorban kifejezhető hétköznapi feladat. A dolgot kézzel kell megcsinálnunk, és mivel a `RENAME TABLE` is hiányzik a megvalósított műveletek közül, ez csak hat lépésben oldható meg:

- Létrehozunk az eredetivel megegyező felépítésű átmeneti táblát.
- Az eredeti tábla tartalmát átmásoljuk ebbe az átmeneti táblába.
- Töröljük az eredeti táblát.
- Létrehozzuk az eredeti nevén az új táblát, új felépítésében.
- Visszamásoljuk az adatokat az eredeti táblából.
- Töröljük az átmeneti táblát.

Érdemes odafigyelni, hova kerül webszerverünkön az sqlite adatbázisfájl. Hasonlóan ugyanis, mint egy rossz helyre tett *inc* állomány, ez is könnyen rossz kezekbe kerülhet. Érdemes tehát azt olyan helyre tenni, amelyet a webszerver közvetlenül nem tud kifelé közvetíteni, de a *PHP* hozzáfér.
Mint a 2. listában is látható, létrehozható ideiglenes, csak a memóriában élő adatbázis is. Ez ellentétben a *MySQL* `HEAP` típusával szemben a létrehozó program futásának végeztével megszűnik létezni. Ha ilyen átmeneti adatbázisra van szükség, fájlnév helyett adjuk fájlnév helyett `:memory:` értéket megnyitáskor.
Az automatikusan növekvő értékű mezők (mint a *MySQL* `AUTO_INCREMENT`-je, vagy a *PostgreSQL* sorozatai) létrehozása *SQLite*-ban roppant egyszerű. Elég, ha a mező az elsődleges kulcs (`PRIMARY KEY`), típusa pedig számszerű, egész (`INT`). Az ilyen mezőknek `NULL` értéket adva azok időben növekvő, egyedi értéket kapnak.
Az *SQLite* nem rendelkezik külön beállításokat tartalmazó állománnyal, telepítése után azonnal üzemkész. Természetesen egyedi finomításokra, beállítgatásokra sokszor szükség lehet. Az ilyen feladatok elvégzésére szolgál az *SQLite* `PRAGMA` parancsa, mellyel egyedi beállításokat érvényesíthetünk, de ez által gyűjthetők be bizonyos információk is.
A `PRAGMA` parancsok a többi *SQL* kéréssel teljesen azonos mód hívhatók. Néhány olyan, amelyre hamar szükség lehet a mindennapi munka során:

```
PRAGMA table_info(tábla_neve)
```

Az adott tábla szerkezetét kapjuk vissza, minden mezőre megkapva az oszlop nevét, típusát, hordozhat-e `NULL` értéket, valamit az alapértelmezett értékét.

```
PRAGMA cache_size = érték
```

Az alapértelmezett 2000 lapos gyorstár méretet bírálhatjuk felül vele. Egy lap mérete körülbelül 1,5 K. Nagyobb értéket adva nő a memóriafogyasztás, de cserébe gyorsabb működésre bírhatjuk lekérdezéseinket bizonyos esetekben.

```
PRAGMA short_column_names=ON/OFF
```

Ellentétben a korábbiakkal, az aktuális *SQLite* változatok több táblás lekérés esetén a a mező nevet a tábla nevével együtt, ponttal elválasztva adják vissza. A fenti kapcsolót `ON`-ra állítva az *SQLite* rávehető, hogy csakis az oszlopnevet adja vissza.

Heilig Szabolcs

KAPCSOLÓDÓ CÍMEK

Michael Owens: SQL adatbázis beágyazása az SQLite segítségével (Linuxvilág, 2003. augusztus, 26-29. oldal)

➲ http://hu.php.net/sqlite

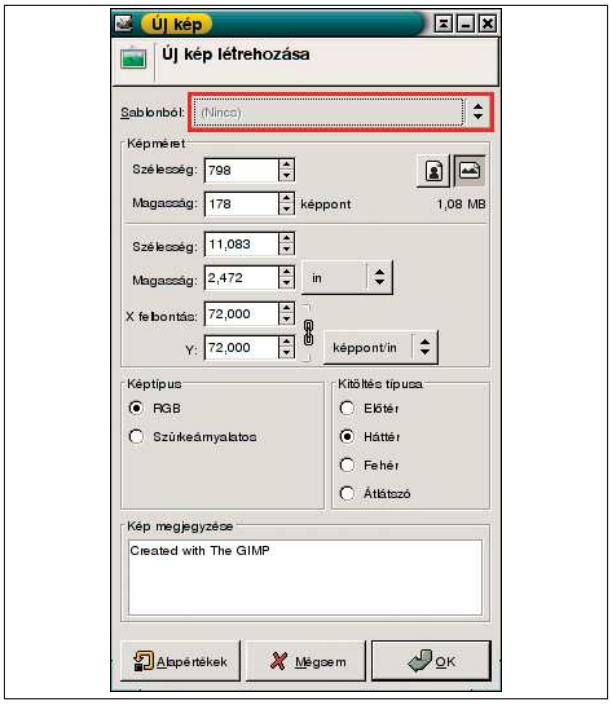
➲ http://sqlite.org

Reklámsáv és logó a GIMP-pel

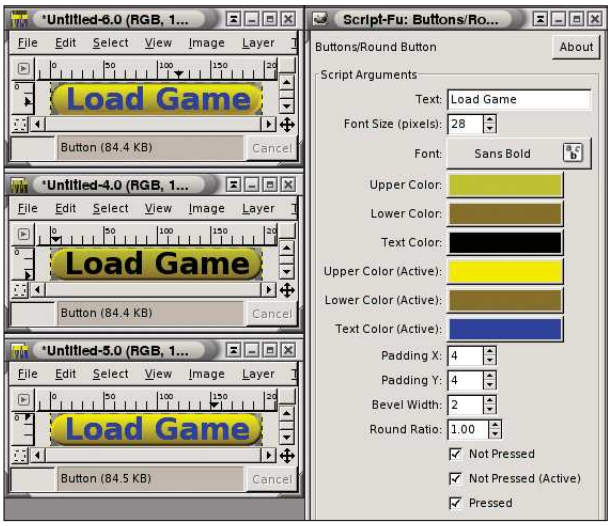
Így az tél közeledtével, mindenki kissé visszavonultabb életet él, tehát bizonyára több ideje van foglalkozni a rajzprogramokkal. Ilyenkor elkezdhetünk gondolkodni a tavaszi újításainkon is. Ilyen lehet például a honlapunkon a reklámcsíkok lecserélése vagy új logó tervezése a cégünk arculatához.

Mindezért nem szükséges nagy pénzeket kifizetni, elegendő csupán elolvasni az alábbi bekezdéseket, majd a *GIMP*-et elindítva máris megkezdhetjük a munkát. Az alábbi példában a képzeletbeli Kuty-Ham Kft. honlapján lévő címsort készítjük el, végeredményül pedig majd az *1. ábrán* látható feliratot kapjuk.
Lássunk is hozzá a feladat megoldásához. Miután elindítottuk a *GIMP*-et, hozzunk létre egy üres képet. Szerencsére a 2.0 változatban már tudunk sablonból is létrehozni képeket. A CTRL+N billentyűk hatására megjelenő párbeszédablakban a 2-es képen pirossal keretezett részen választhatunk a különféle sablonok közül. Válasszuk ki a ***Web banner Huge*** változatot, majd adjunk meg nagyobb magasságot.
A kép létrehozása után folytathatjuk a munkálatokat a szöveg elkészítésével.
Válasszuk ki a dinamikus szövegek létrehozására alkalmas eszközt, melyet ***T*** betű formájú ikonjáról ismerhetünk fel. Kattintsunk valahová a képen, ahol feltehetően kezdődni fog a felirat és az eszközbeállításoknál válasszuk ki a megfelelő betűtípust és méretet. A kis ablakban a szövegmezőbe írjuk be a felirat szövegét. Zárjuk be a párbeszédablakot, majd a létrehozott szöveget helyezzük el a megfelelő helyre a rétegek mozgatására szolgáló eszközzel. Ezt az eszközt a négy irányba mutató nyíl jelzi. Talán kezdetben kicsit nagyra választottuk a képméretet, de ez nem okozhat gondot kedves olvasóimnak, hiszen a kép levágására is mutattam eszközöket az előző részekben. Alakítsuk ki a megfelelő méretet, de vigyázzunk arra, hogy az árnyéknak is kell majd hely, tehát nem szabad pontosan levágni a karakterek szélénél a képet.
A méret meghatározása után következhet a színezés. Válasszuk ki a színek alapján a fekete képpontokat. Az új *GIMP*-ben már az eszköztáron is megtalálható a szín szerinti kiválasztásra alkalmas eszköz, amit egy színes négyzetekre mutató kéz jelez. Válasszuk ki tehát a fekete színt a képről, majd készítsünk átmenetet. Keressük meg a cég arculatához leginkább illeszkedő átmenetet vagy készítsünk tetszőleges, új átmenetet a *GIMP* segítségével. Mindenkinek saját ízlése szerint kell eljárnia a következő lépéseknél, de ennek ellenére én is leírom az általam alkalmazott megoldást. A ***Golden*** elnevezésű átmenet kiválasztása után beállítjuk az átmenet típusát, ami jelen esetben a ***Gömbölyű shapebrust***. Mivel a kijelölés illeszkedik a betűk formájához, elég csak egy rövid, vízszintes részen megadni a színátmenet irányát, hogy a képen látható hatást elérjük.
A festés után következzen az árnyék létrehozása, melyet egy beépülő modullal készíthetünk el. A kijelölés maradjon változatlan és a kép menüjében a ***Script-Fu*** menüpontból a ***Shadow->Drop Shadow*** pontokat választva készíthetjük

1. ábra A cél: új címsorunk elkészítése



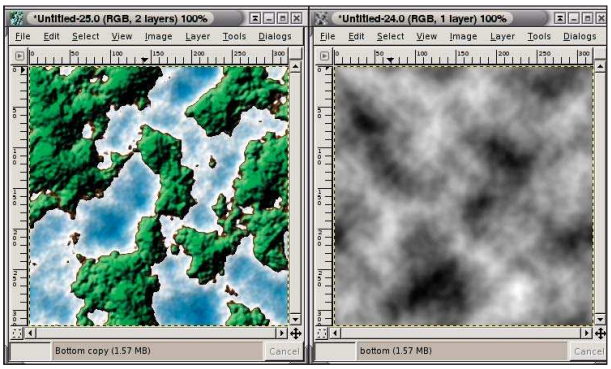
2. ábra Új kép sablonból



3. ábra Gombok beépülő modul segítségével

4. ábra Feliratok beépülő modulokkal

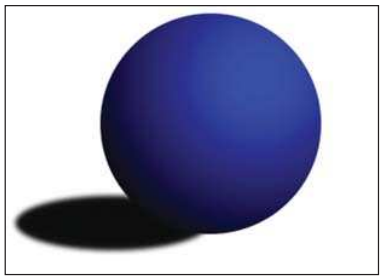
el az árnyékot. Az alapbeállítások (8 képpontnyi eltolás mindkét tengely irányában) megfelelőek, de tetszés szerint meg is változtathatjuk őket. Miután létrehoztuk a vetett árnyékot, következhet a betűkön látható körvonal, melyet szintén színátmenettel tudunk kitölteni. A kijelölést most alakítsuk keretté, mégpedig a kép menüjében található ***Kijelölés->Keret*** menüpontok kiválasztásával. Adjunk meg tetszőleges értéket, a példában három pixeles keretet használtam. Így újra alkalmazhatjuk a színátmenettel történő kitöltést. Ezzel a lépéssel már a felirat rendben van, tehát foglalkozhatunk a háttérrel és a különféle egyéb hatásokkal.
A háttér réteget válasszuk ki a rétegek kezelésére szolgáló ablakban, majd hozzunk létre egy új réteget fehér színnel. Az újonnan létrejött réteget töltsük ki valamilyen szürkéskék árnyalattal, de itt sem szabad határt szabnom az olvasók kreativitásának, tehát mindenki a neki megfelelő színnel töltheti ki a réteget. A réteg átlátszóságát vegyük körülbelül 50 százalékosra, majd a megjelenítés módját állítsuk ***Szorzás***-ra. Ez utóbbi meghatározható az átlátszóság megadására szolgáló csúszka feletti listában. Miután elvégeztük a színezésre szolgáló réteg beállításait, válasszuk ki a legalsó 'Háttér' réteget, és hozzuk létre a mintázatát. A kép menüjében a ***Filters->Artistic->Apply Canvas*** menüpontokon keresztül hozhatunk létre vászon mintázatot, vagy ha magyar nyelv van beállítva a *GIMP*-ben, akkor a ***Szűrők->Művészi->Apply Canvas*** menüpontok kiválasztásával. Adjuk meg a megfelelő értékeket és hozzuk létre a vászonmintázatot.
A vászon színét többféleképpen is meghatározhatjuk.
A másik megoldás az lehet, hogy nem készítünk félig átlátszó szűrőréteget, hanem helyette kiválasztjuk a háttér réteget és a kép menüjében a ***Réteg->Színek->Colorize*** menüpontokon keresztül megjelenő párbeszédablakban meghatározzuk a vászon színezetét, telítettségét és világosságát.
Amint látható, ***Linux*** eszmeiségéhez méltó programmal találkoztunk, a feladatok megoldása során több utat is bejárhatunk, újabb és újabb ismeretekhez juthatunk általuk. Az alábbiakban más megoldásokat is mutatok a feliratok elkészítésére, bevezetve ezzel kedves olvasóimat a *GIMP* bővítéseinek lehetőségeibe.



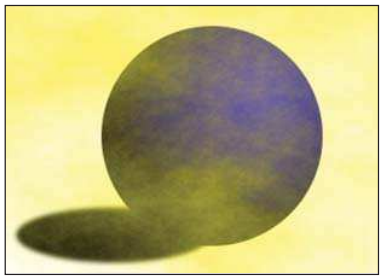
5. ábra „Nékem térkép e táj..."

A program egyik nagy előnye, hogy magunk is írhatunk beépülő modulokat, mindazon függvények használatával, melyre a *GIMP* rutinjai lehetőséget adnak. Az újabb változatban már nem csupán a bonyolult formával rendelkező ***Scheme*** programnyelven készíthetünk ilyen kiegészítőket, hanem az egyik legkönnyebben megtanulható programnyelven is, ***Python*** programokat írhatunk. Így azok is élvezhetik a bővítés előnyeit, akik mindeddig nem tanultak semmilyen programozási nyelvet, mivel a ***Python*** egyszerűsége ellenére egy hatékony eszköz a kezdők kezében is. A képek elkészítése során gyakran találkozhatunk olyan ismétlődő feladatokkal, melyeket érdemes rövid kis programokban összefoglalni. A következő hónapokban ilyen programokkal ismerkedhetünk meg, betekinthetünk elkészítésük menetébe. Most azonban ízelítőül lássunk néhány példát, miről is lesz majd szó.
A beépülő modulok mindegyikében meghatározható, hogy a program menürendszerében melyik almenüben kapjanak helyet, így az eddigiek során is találkozhattunk már velük. Amikor azonban nincs megnyitva egyetlen kép sem, a *GIMP*-ben ilyenkor is elérhetünk bizonyos beépülő modulokat. Jellemzően ezek a modulok új képek előállítására szolgálnak. Tekintsük meg az eszköztár menüjében az ***Xtns->Script-fu*** menüpontokat. Látható, hogy a kiinduló képet nem igénylő modulok között számos lehetőséget találunk például gombok

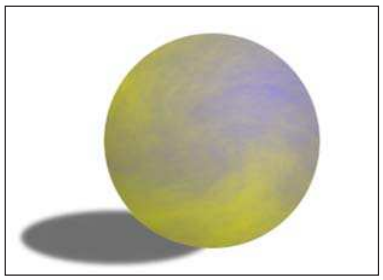
vagy feliratok elkészítésére. Ilyen gombokat láthatunk a 3-as képen is, most azonban válasszuk ki, a menüből a ***Buttons*** almenüt, és készítsünk magunk is egy tetszésünknek megfelelő gombot. A gombokat felhasználhatjuk majd játékunkban, honlapunkon vagy ízlésünknek szerint más helyen.
A képen látható gombok elkészítéséhez válasszuk ki a ***Xtns->Script-Fu->Buttons->Round Button*** menüpontokat és a megjelenő párbeszédablakban írjuk be a kívánt szöveget. Itt állíthatjuk be, hogy milyen színekkel készüljön a gomb, választhatunk betűtípust, betűméretet, megadhatjuk a gomb feliratát és meghatározhatjuk, hogy milyen állapotokhoz készítsen a program képeket. Egy gomb általában három állapotú lehet, lenyomott, ki nem választott és kiválasztott. Ebben a beépülő modulban kiválaszthatjuk a szükséges állapotokat, majd az ***OK*** gombra kattintva máris elkészül a megadott beállításoknak megfelelő gomb.
Sokan a *GIMP*-et honlapjuk díszítéseinek elkészítésére használják, vagyis sok esetben szükség lehet feliratok készítésére. A beépülő modulok között szép számmal akadnak ilyen feliratok előállítására alkalmas programok, melyeket az eszköztáron a ***Xtns->Script-Fu->Logos*** menüpontok kiválasztása után tetszés szerint kipróbálhatunk, tesztelhetünk és kiválaszthatjuk az adott feladathoz legjobban illőt. Érdemes próbálkozni a különféle színbeállításokkal, hiszen a próbálgatás hozhatja meg a megfelelő tapasztalatot, így választhatjuk ki majd a szívünkhöz és gondolkodásunkhoz legközelebb álló változatot. Ilyen feliratokra látható néhány példa a négyes képen.



6. ábra Kiindulási kép a varázsgömbhöz



7. ábra Már alakul...



8. ábra Íme a végeredmény

Haladjunk tovább a menüben és lássuk mire képes még ez a programnyelv és a *GIMP*. A beépülő modulok között találhatunk különféle mintázatok előállítására alkalmas darabokat is. Ilyenek például az 5-ös képen látható térképszerű tájak előállítására alkalmas ***Land*** és a ***Flat Land*** amelyek szintén jó szolgálatot tehetnek a játékunk- vagy térképészeti alkalmazásunk reklám oldalának elkészítése során.
Sok esetben szükségünk lehet arra, hogy egy szöveges dokumentumban található, előre elkészített szöveget helyezzünk el a képeinken. A ***Xtns->Script-Fu->Utils->ASCII To Image*** menüpontok kiválasztásával megadhatunk egy ilyen szöveges állományt, majd a beépülő modul segítségével a *GIMP* előállítja a szöveget tartalmazó képet. Ebben az esetben a *GIMP* tulajdonképpen minden szövegsorhoz új réteget hoz létre, tehát az egyszerűbb kezelés érdekében célszerű a szöveget tartalmazó rétegeket egyesíteni.
Amint korábban már említettem, a másik lehetőségünk a külső programok írásához a ***Python*** programozási nyelv elsajátítása és használata. Előfordulhat, hogy a terjesztésünk nem tartalmazza a ***Python-Fu*** modult, ebben az esetben az Internetről letölthetjük majd saját igényeinknek és számítógépünknek megfelelően forráskódból telepíthetjük. Sajnos egyenlőre még fiatal lehetőségről van szó, ami azt jelenti, hogy pillanatnyilag nincs túl nagy választék a ***Python-Fu*** használatával készült modulok között, de mindenesetre ígéretes vállalkozásnak látszik. Mindezt amiatt gondolhatja az ember, mivel a ***Script-Fu*** kiegészítők kínálata az évek során nem változott jelentősen, talán éppen azért, mert a ***Scheme*** nyelv nem tartozik a legegyszerűbben elsajátítható programozási nyelvek közé. A későbbiekben látni fogjuk majd, hogy a zárójelek tömegében még a gyakorlott szem is könnyedén eltéved. Jómagam azt gondolom, hogy kezdetben nem szenvedünk semmilyen hátrányt az által, ha a ***Python*** modulokkal ismerkedünk meg.
Ízelítőül lássunk néhány ilyen példát az legalapvetőbb ***Python*** programok igénybevételével és tekintsük meg a mellékelt képeket. Hozzunk létre kiindulási alapként egy árnyékolt gömböt. Válasszuk ki a ***Xtns->Pyhton-Fu->Test->Sphere*** menüpontokat, majd állítsuk be a gömb színét valamilyen tetszőleges kék színre. Ezután következzen a ködösítés, vagyis a kép menüjében válasszuk ki a ***Python-Fu->Effects->Add Fog*** menüpontokat és a köd színét válasszuk citromsárgára. A hatás létrehozása után még egy ***Python*** eszközt használunk, mely a ***Python-Fu->Distorts->Whirl and Pinch*** nevet viseli és egyfajta örvénylő mozgást jelenít meg a kiválasztott rétegen. Jelenleg ha nem változtattunk semmit, akkor a kiválasztott rétegünk a ködöt jeleníti meg, tehát a szűrő alkalmazása után a sárga köd örvényleni látszik. Az utolsó előtti lépés a hetes képen látható gömb előállításához a rétegműveletek megváltoztatása. A rétegek kezelésére szolgáló ablakban állítsuk be a ködöt megjelenítő réteget, hogy a *GIMP* csak a színeket vegye figyelembe, vagyis válasszuk ki a ***Color*** típust. Végezetül készítsünk egy rétegmaszkot, ami csak a gömbön engedi megjelenni a sárga ködöt. Jelöljük ki a gömböt, majd a köd rétegén jobb gombbal kattintva válasszuk ki a ***Add Layer Mask*** lehetőséget. A maszk forrása legyen a kijelölt terület (***Selection***). Ezzel a néhány lépéssel – melyeket nyomon követhetünk a 6-es ábrától kiindulva –, láthatóan kevés kézzel végzett művelettel elkészíthetjük 'varázsgömbünket'.
Ebben a hónapban ennyi fért a *GIMP*-ről szóló bemutatómba, a következő lapszámokban azonban részleteiben is ismertetem a ***Python*** beépülő modulok készítésének lehetőségeit. Bízom benne, hogy ezáltal mindenki számára világossá válik: a programírás nem nagy kunszt, csupán elhatározás és néhány száz oldalnyi jófajta szakirodalom elolvasása kell hozzá.

Fábián Zoltán

A pingvin igenis vándormadár (3. rész)

Avagy használjunk Linuxot a mobil eszközökön.

Sorozatomnak ebben a részében megvizsgálunk néhány „laptop specifikus” X beállítást, továbbá olyan felhasználói programokat, amelyeknek minden notebook felhasználó nagy hasznát veheti.

Hogyan állítsuk be az X-et?

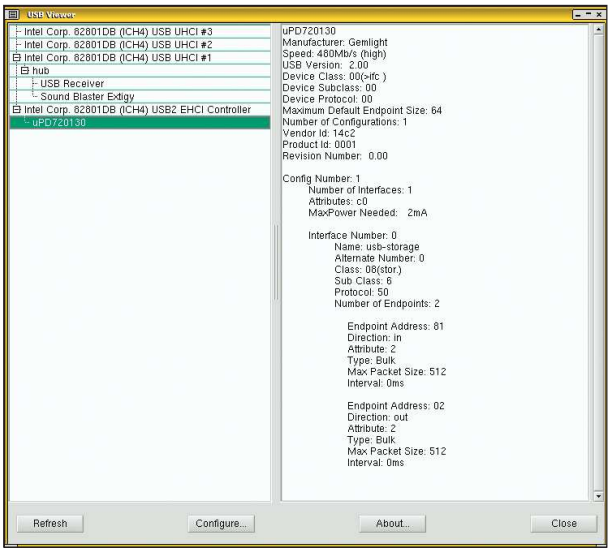
Aki telepített már asztali gépre ***Linuxot*** és hozzá grafikus felületet, az most némiképp joggal gondolhatja úgy, hogy ez triviális. Én is azt hittem egészen addig, amíg közelebbi barátságba nem keveredett a notebookom és a ***Linux***. A telepítés maga tényleg teljesen hasonlóan zajlott, mint a régi asztali gépemnél, de amint végeztem, máris jöttek a problémák. A videókártya meghajtóprogramja valahogy nem volt az igazi. Nem volt ***OpenGL*** támogatás, a touchpad érintésére az egér kurzor őrült táncba kezdett és még sorolhatnám... Most ezeket a problémákat és megoldásukat szeretném bemutatni, hátha valakit ezzel egy több napos fejfájástól tudok megkímélni.

A felbontás beállítása

Az ***xfree86*** csomagok telepítése közben elindul egy szkript, amelynek segítségével be tudjuk állítani a notebookunk kijelzőjét. Először is jelöljük meg, hogy ***LCD*** képernyőt használunk, aztán folytassuk a kijelző paramétereink megadásával.

A legújabb notebookok némelyike már gyönyörű nagy felbontású kijelzőkkel rendelkezik, ám például az ***1400x1050***-es felbontás olyan messze áll a ***VESA*** szabványtól, mint a ***Microsoft*** a nyílt forráskód támogatásától. Ez bizonyos modellek esetében komoly problémát jelenthet, ugyanis egyszerűen képtelenek leszünk normális felbontással használni a gépünket. Ilyen például az ***Acer Travelmate 600***-as sorozat némelyik tagja. Ezeknél a gépeknél ugyanis a ***BIOS*** nem igazolja vissza az ***X***-nek a kért felbontást, így az ***X*** nem indul el ebben a felbontásban. Ugyan használhatjuk a gépünket ***1280x1024***-es felbontásban, ám jó ha tisztában vagyunk vele, egy ***LCD*** képernyőt ha nem a fizikai felbontásában használunk, akkor jelentősen romlik a kép minősége, ugyanis a képernyő ennél a technológiánál tényleg fizikailag képpontokból áll. Egy-egy képpont egy-egy ***LED*** hármasból (piros, zöld, kék ***LED***) áll, így ha fél képponttal kéne csúsztatni, akkor egy ***LED***-nek két fél képpontot kéne megjeleníteni, ami fizikai képtelenség. A régi kijelzők ebben az esetben vagy levették a megjelenítendő területet akkorára, amekkora felbontásra állítottuk, vagy irtózatosan ronda képet adtak vissza. A mai kijelzőket már valamennyire felkészítették ennek a problémának a kezelésére, így használnak elmosást, de a tökéletestől még így is nagyon-nagyon messze állunk.

1. kép A cacrdinfo program felhasználói felülete – Flash memóriakártyával a foglalatban



2. kép Az usbview program és az általa szolgáltatott információk

Amennyiben a választható felbontások között nem találjuk azt a felbontást, amelyet a monitorunk tud, akkor még mindig nincs minden veszve. Az ***/etc/X11/XF86Config-4*** állo-

mányban megpróbálhatjuk kézzel megadni a felbontást. Ezt a `Screen` szekcióban tudjuk általában megadni. Álljon itt egy példa, az én gépem beállításai. Látszik, hogy a felbontások közé bekerült az ***1400x1050***-es felbontás.

```
Section "Screen"
        Identifier      "Default Screen"
        Device          "Generic Video Card"
        Monitor         "Generic Monitor"
        DefaultDepth    24
        SubSection "Display"
                Depth           1
                Modes           "1400x1050"
➥"1280x1024" "1280x960" "1152x864" "1024x768"
➥"800x600" "640x480"
        EndSubSection
        SubSection "Display"
                Depth           4
                Modes           "1400x1050"
➥"1280x1024" "1280x960" "1152x864" "1024x768"
➥"800x600" "640x480"
        EndSubSection
        SubSection "Display"
                Depth           8
                Modes           "1400x1050"
➥"1280x1024" "1280x960" "1152x864" "1024x768"
➥"800x600" "640x480"
        EndSubSection
        SubSection "Display"
                Depth           15
                Modes           "1400x1050"
➥"1280x1024" "1280x960" "1152x864" "1024x768"
➥"800x600" "640x480"
        EndSubSection
        SubSection "Display"
                Depth           16
                Modes           "1400x1050"
➥"1280x1024" "1280x960" "1152x864" "1024x768"
➥"800x600" "640x480"
        EndSubSection
        SubSection "Display"
                Depth           24
                Modes           "1400x1050"
➥"1280x1024" "1280x960" "1152x864" "1024x768"
➥"800x600" "640x480"
        EndSubSection
EndSection
```

Ha végeztünk a beállításokkal, akkor indítsuk újra az ***X***-et (például a CTRL+ALT+BACKSPACE billentyűkombinációval) és amennyiben elindul az ***X*** hiba nélkül, akkor legyünk nagyon boldogok. Amennyiben nem, akkor megint érdemes a nethez fordulni. Nézzünk körül a ***tuxmobile.org***-on, vagy a különböző fórumokban, hogy másoknak milyen tapasztalatai vannak az adott modellel.

OpenGL

A következő probléma a videókártyámmal az volt, hogy az ***X***-hez adott ***Radeon*** meghajtó úgy tűnt mintha nem akarta volna támogatni a hardveres ***OpenGL*** gyorsítást. Így minden ***OpenGL***-es programom enyhén szólva lassú volt. Kis keresgetés a neten és világossá vált számomra, hogy a meghajtó amit használok valóban nem támogatja a hardveres ***OpenGL***-t. továbbá a grafikus eszközök gyártói a mobil eszközökbe épített lapkák terméktámogatását általában az adott eszköz forgalmazójára hárítják, sajnos azonban ezek a forgalmazók pedig nem igazán nyújtanak támogatást a Linuxhoz. Már kezdtem szomorkodni, hogy a ***Radeon 7500 Mobility*** kártyámmal sikerült rossz lóra tennem, de aztán jött a reménysugár. Úgy hívják, hogy ***xserver-xfree86-dri-trunk***. Ezt a csomagot kell feltennünk és beállítanunk, hogy kihasználhassuk a kártyánk nyújtotta 3D gyorsítási lehetőségeket. A csomag nincs benne a szokásos ***Debian*** forrásban, így a forráslistánkat ki kell egészítenünk a

```
deb http://dri.freedesktop.org/~daenzer/debian/
➥dri-trunk-sid/ ./
```

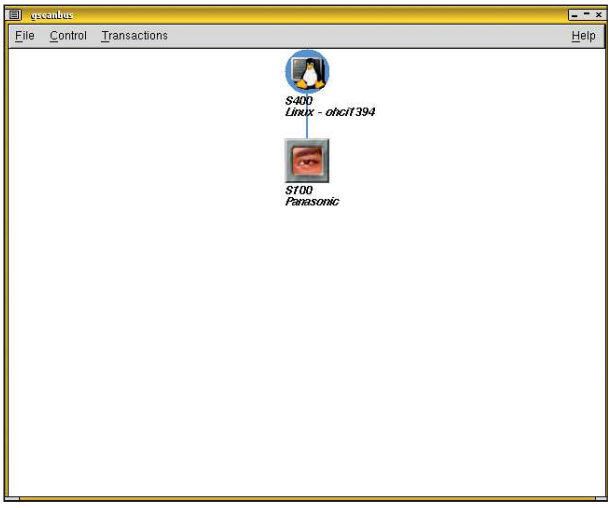
sorral. Ezután jöhet az

```
apt-get update
```

és telepíthetjük is a csomagot.
A telepítés folyamán a csomag beállításra is kerül, így sok teendőnk nincs vele. Amennyiben mégis problémánk merülne fel, akkor olvassuk el a ***/usr/share/doc/xserver-xfree86-dri-trunk*** könyvtárban található leírást.
Ha beállítottuk, akkor a `glxgears` nevű programmal tesztelhetjük a kártyát. Ez a program frissítési értékeket számol, így az ***FPS (frame/second)*** vadászok rögtön tudni fogják mit bír a kártya az adott felbontással. Aki pedig élesben is kipróbálná a kártyát, annak ajánlom a ***tuxracer*** nevű játékot. Itt szeretném megjegyezni, hogy az ***Nvidia*** és az új ***ATI*** kártyákhoz már letölthető a gyári meghajtóprogram a gyártók weboldaláról. Ezekkel a meghajtókkal tapasztalataim szerint nem szokott semmilyen probléma előfordulni a 3D gyorsítás terén.

TV és külső monitor kimenetek kezelése

A mobil eszközökbe a múltban és jelenleg is előszeretettel használnak a gyártók ***ATI*** grafikus processzorokat. Régebben a ***Rage***, újabban a ***Radeon*** lapkák a népszerűek. Ezek a kártyák minden esetben tartalmaznak külső monitor csatlakozót és a többségükön már a TV kimenet is megtalálható. Ahhoz, hogy ezeket a kimeneteket használni tudjuk, szükség lehet arra, hogy egy külső alkalmazás segítségével engedélyezzük és beállítsuk őket. TV kimenet esetén például szükség lehet a használni kívánt TV szabvány beállítására. ***ATI*** kártyák esetén egy ügyes kis alkalmazás erre a célra az ***atitvout*** csomag tartalma. Miután telepítettük a programot az ***atitvout*** paranccsal tudjuk futtatni. A parancs beírása után kapunk egy rövid használati utasítást, amelyben le van írva, hogy miként tudjuk az egyes kijelzőket, illetve azok csoportjait aktiválni, vagy kikapcsolni, miként tudjuk a TV kimenetet életre kelteni és a szabványt beállítani. Megjegyezném azonban, hogy a program sajnos nem száz százalékosan működik, az én ***ATI Mobility Radeon 7500***-as kártyám ***S-Video*** kimenetét ugyanis nem sikerült vele beállítani, ugyanakkor ugyanezen ***7500***-as kártya kompozit kimenetes változata minden probléma nélkül beállítható.



3. kép A gscanbus felhasználói felülete

Ugyan a TV kimenetem még nem működik, de a külső monitor csatlakozó elsőre működött, nem is akárhogy. A gépem kijelzője egy ***1400x1050***-es ***TFT*** panel és a külső csatlakozóra egy ***1280x1024***-es ***TFT***-t kötöttem rá. És ami az ***X*** indítása után történt, az meglepett. ***Windows*** alatt egy ilyen helyzetben a kisebbik képernyőn az asztal területe nem fér el, így az egeret mozgatva az asztalon a kép csúszkál ideoda, attól függően, hogy az asztal melyik területén tartózkodom. ***X*** alatt pedig az ***1400x1050***-es méretű asztalom egy huszárvágással le lett kicsinyítve ***1280x1024***-re. Ez alatt azt kell érteni, hogy a notebookom kijelzője továbbra is ***1400x1050***-es felbontással üzemelt, míg a külső ***TFT*** monitor úgy működött ***1280x1024***-es felbontásban, hogy közben a teljes munkafelületem látszott a képen. Ugyan nem volt penge éles a kép, de teljesen használható volt. Ennek igazából akkor vehetjük hasznát, ha egy projektort kötünk a kimenetre, ugyanis anélkül, hogy állítgatni kéne a felbontást tudunk rendes képet vetíteni. Ismét le a kalappal a ***Linux*** előtt.

A notebook gyártók által előszeretettel használt másik grafikus lapka az ***Intel Extreme chipset***. Ahhoz, hogy ezeken a grafikus kártyákon engedélyezzük a külső monitor kimenetet az i810switch csomagra lesz szükségünk. Ha telepítettük, akkor az `i810rotate` paranccsal tudjuk kapcsolni, hogy melyik kijelző legyen aktív. A három lehetőség a csak ***LCD***, ***LCD*** és ***CRT***, csak ***CRT***. Ezeken a variációkon tudunk lépkedni a parancs futtatásával. Amennyiben ezt a parancsot hozzárendeljük az Fn+F5 kombinációhoz (lásd a ***Gyorsbillentyűk beállítása*** című részt), akkor a ***Windows***-os működéshez teljesen hasonló működést kaphatunk.

A touchpad beállítása

Miután telepítettük a gépünkre az ***X***-et és valamilyen ablakkezelő rendszert, boldogan indítjuk is a grafikus felületet, ám hamar ismét üröm vegyülhet örömünkbe. Hozzáérünk az egérhez, amire az elkezd össze-vissza ugrálni, teljesen értelmetlenül kattintgat, hol a bal, hol a jobb gombbal. A probléma abból adódik, hogy a tapipad kicsit más protokoll szerint adja a jelet a gépnek, mint a szabvány ***PS/2***, ***ImPS/2*** egerek. A megoldás? Természetesen telepíteni kell a megfelelő csomagot az ***X***-hez és elvégezni a beállításokat a már említett ***XF86Config-4*** állományban. Először tehát telepítsük a ***xfree86-driver-synaptics*** csomagot, majd az ***/usr/share/doc/xfree86-driver-synaptics*** könyvtárban található ***README.Debian*** állományban található példa alapján készítsük el a beállításainkat az ***XF86Config-4*** állományban. Én az alábbi beállítást használom:

```
Section “InputDevice”Section “InputDevice”
        Driver          “synaptics”
        Identifier      “Touchpad”
        Option          “CorePointer”
        Option          “Device”
“/dev/psaux”
        Option          “Protocol”      “auto-dev”
        Option          “LeftEdge”      “1700”
        Option          “RightEdge”     “5300”
        Option          “TopEdge”       “1700”
        Option          “BottomEdge”    “4200”
        Option          “FingerLow”     “25”
        Option          “FingerHigh”    “30”
        Option          “MaxTapTime”    “180”
        Option          “MaxTapMove”    “220”
        Option          “VertScrollDelta” “100”
        Option          “MinSpeed”      “0.06”
        Option          “MaxSpeed”      “0.12”
        Option          “AccelFactor” “0.0010”
EndSection
```

Fontos, hogy ne felejtsük el a `Touchpad` eszközt betölteni a `ServerLayout` szekcióban. Ez az alábbi példa alapján tehető meg:

```
Section “ServerLayout”
        Identifier      “Default Layout”
        Screen          “Default Screen”
        InputDevice     “Generic Keyboard”
        InputDevice     “Touchpad”
        InputDevice     “Generic Mouse”
EndSection
```

Fenti beállítással a tapipadunkat beállítottuk, méghozzá nem is akárhogy. A pad jobboldali szélén húzva ujjunkat görgethetjük az ablakot fel, illetve le, ugyanezt az alsó szélén csinálva horizontális gördítést csinálhatunk. Normál bal klikket egy ujjal ütve tudunk csinálni, középső gomb két ujj, jobb klikk három ujj. A pad sarkaiba tapintva további funkciók érhetők el. A jobb alsó sarok szintén jobb klikk, míg a bal felső kijelölt szövegre vonatkozó beillesztés művelet.

Gyorsbillentyűk beállítása

A mai notebookok szinte mindegyikén találhatóak már különböző gyorsbillentyűk, amelyeket felprogramozva gombnyomásra indíthatjuk a kedvenc böngésző és levelezőprogramunkat, bekapcsolhatjuk a vezeték nélküli hálózati kártyát, vagy elindíthatjuk a ***CD*** lejátszást. Ahhoz, hogy ezeket a jópofa funkciókat kihasználhassuk több beállítási mód közül is választhatunk. Elsőként az okosabb ablakkezelők, mint amilyen a ***Gnome2***, vagy a ***KDE*** már tartalmaz-

nak beépített kezelői felületet ezeknek a funkcióknak a megvalósítására. *Gnome 2.6* alatt a forróbillentyűk beállításainál bizonyos funkciókhoz hozzá tudjuk rendelni nem csak a különböző billentyűkombinációkat, hanem ezeket az extra gombokat is.

Másfelől bizonyos alkalmazások tartalmaznak olyan beépülő elemeket, bővítményeket, amelyek szintén lehetővé teszik ezen gombok használatát. Ilyen például az *XMMS*-hez tartozó *XF86Audio Keys Control plug-in*.

Harmadik és talán a leguniverzálisabb megoldás egy olyan felhasználói alkalmazás telepítése, amellyel ezekhez a billentyűkhöz egyedi parancsokat tudunk rendelni. Ilyen alkalmazást például az *xhkeys* csomag tartalmaz. Ha telepítettük a csomagot, akkor utána készíthetünk a teljes rendszerre érvényes beállításokat, de készíthetünk felhasználónkénti egyedi beállításokat is. Mindezt úgy tehetjük meg, hogy futtassuk az `xhkconf` parancsot, üssünk le egy billentyűt, amelyhez funkciót kívánunk társítani, majd után adjuk meg, hogy milyen eseményt szeretnénk a billentyűhöz társítani. Ez lehet egy alkalmazás indítása, egy belső funkció elérése, egy billentyűzet, vagy egér esemény. Ha befejeztük a billentyűk beállítását, akkor hagyjuk kicsit magára a programot, ugyanis a konfigurációk elmentése és a beállítás befejezése akkor történik meg, ha tíz másodpercig nem nyúlunk a billentyűzethez. Ha végeztünk a beállításokkal, akkor futtassuk az `xhkeys` parancsot, ezzel aktiválva a beállításainkat. Ha azt szeretnénk, hogy a grafikus felület minden indításánál automatikusan induljon a forróbillentyű kezelőnk is, akkor érdemes az *X* folyamatba (session) elmenteni.

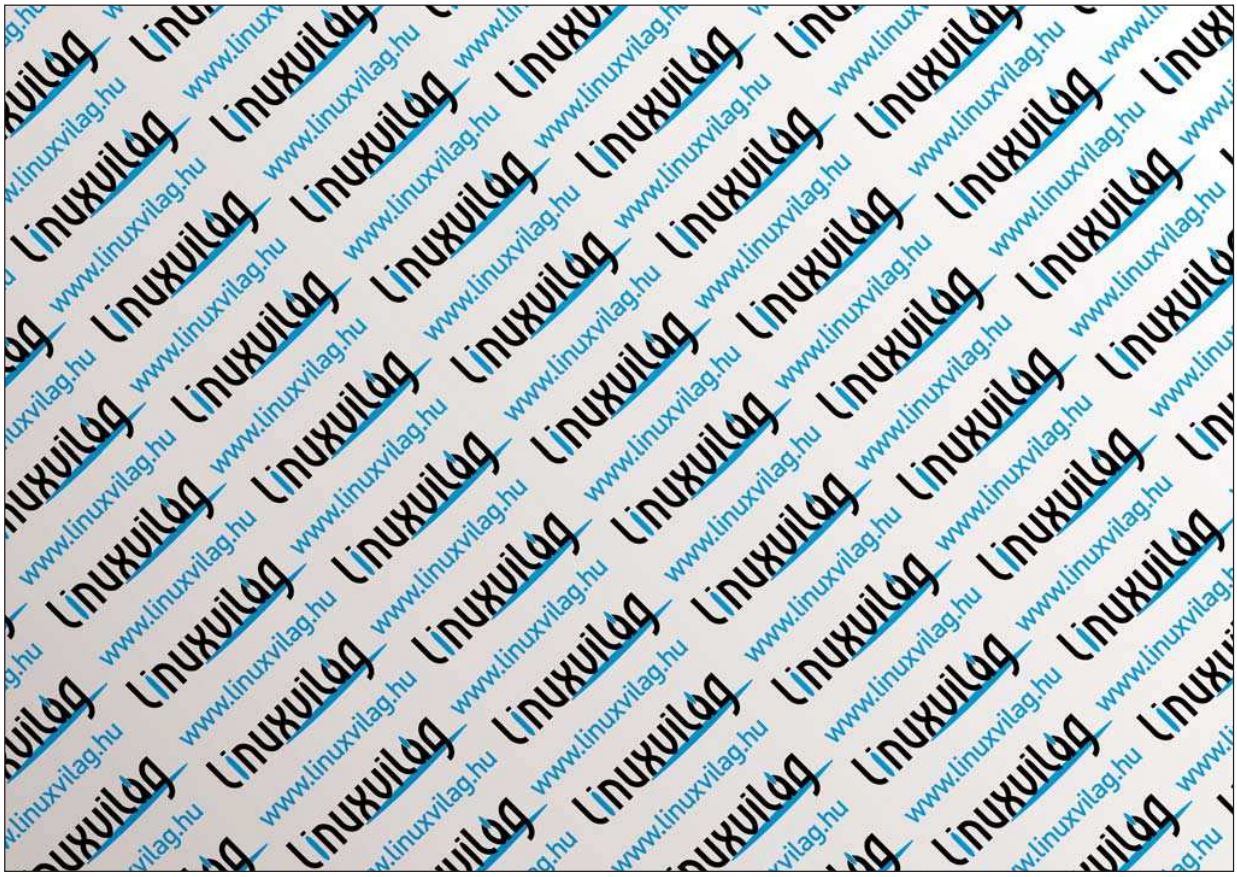
A következő három bekezdésben megnézzük, hogy milyen grafikus felügyeleti megoldások vannak a *PCMCIA* kártyák, illetve a géphez csatlakoztatott *USB* és *FireWire* eszközökhöz.

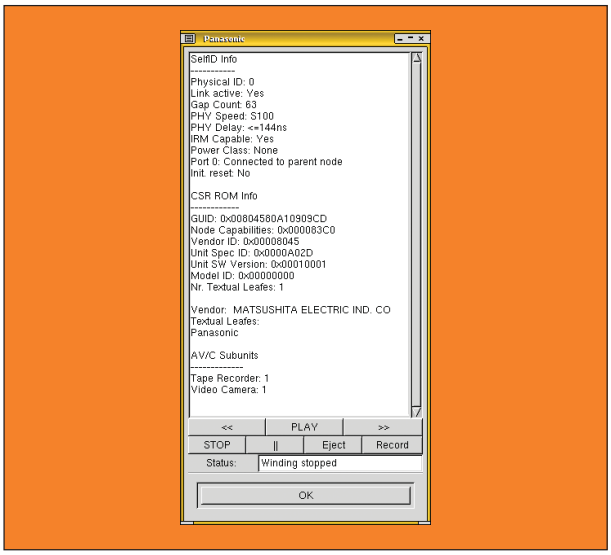
A PCMCIA kártyák felügyelete

Amennyiben a gép telepítésekor nem töröltük a *pcmcia-cs* csomagot és a rendszermagban a támogatás is engedélyezve van a *PC*-kártyákhoz, akkor nincs különösebb teendőnk a *PCMCIA* kártyák működésre bírásához. Amennyiben bármelyik fenti feltétel nem teljesül, akkor végezzük el a szükséges beállításokat, telepítsük a *pcmcia-cs* csomagot.

Ha ezzel megvagyunk, akkor dugjunk be egy kártyát a csatlakozóba. A `cardctl` parancsot megfelelően paraméterezve konzol alól tudjuk kezelni a kártyákat. Amennyiben ugyanezt a munkát egy szép grafikus programból szeretnénk elvégezni, akkor használjuk a `cardinfo` nevű programot. Ezt a programot indítva egy aranyos kis grafikus programot tudunk futtatni, amellyel alapinformációkat kaphatunk az egyes nyílásokba helyezett kártyákról, valamint kezelni tudjuk őket. A kártyákat fel tudjuk függeszteni, visszatölteni, vagy éppen eltávolítani a rendszerből. Ez azért lehet fontos, mert egy olyan kártyának az eltávolítása, amelyet a rendszer még nem engedett el könnyen kernel pánikhoz vezethet.

Azok, akik rendelkeznek *PCMCIA* felületű kártyaolvasóval, adapterrel, azok nagy hasznát vehetik annak, hogy *Linux* alatt a megfelelő rendszermagmodul betöltése esetén a *PCMCIA* foglalatba helyezett memóriakártyák szabvány *IDE* eszközként kezelhetőek. Ehhez mindössze a rendszermag beállításainál a *Device Drivers/ATA-ATAPI Devices/*





4. kép A gscanbus által szolgáltatott információk

PCMCIA ATA Support modult kell a rendszermagba fordítani. Utána betoljuk a kártyát a helyére és a `cardinfo` rögtön meg is mondja, hogy melyik */dev* eszközként tudjuk becsatolni a meghajtót.

Az USB csatlakozók és eszközök felügyelete

Megfelelően beállított rendszermag esetén az *USB*-re kötött különböző eszközök elvileg azonnal használhatóvá válnak, a csatlakozási felületük megjelenik a */dev/usb* könyvtárban, az eszközök listázhatóak az `lsusb` paranccsal. Ha szeretnénk egy látványos topológiát is megjeleníteni, valamint további információkra is szükségünk lenne az *USB*-s perifériákkal kapcsolatban, akkor használjuk az *usbview* csomag `usbview` parancsát. Ezzel egy egyszerű grafikus alkalmazást indítunk el, amely megjeleníti a teljes *USB* fát, amely a gépünkhöz kapcsolódik. Az egyes eszközöket kijelölve a jobboldali ablakban olyan fontos információk jeleníthetőek meg, mint az eszköz által foglalt sávszélesség, az *USB* csatlakozás sebessége, a gyártó neve, a pontos modellszám. Ezek nagyon hasznos információk lehetnek egyfelől a lehető leghatékonyabb erőforráskihasználás eléréséhez, másfelől hibakeresés, vagy meghajtó program telepítésekor is nagy hasznát vehetjük ezeknek az információknak.
Notebookok esetén sokszor felmerül az igény, hogy *USB*-re különböző külső meghajtókat csatlakoztassunk. Ezt könnyen megtehetjük, ha a rendszermagban engedélyezzük az *USB Storage Support* modult. Ezzel a modullal egyszerűen csatlakoztathatunk különböző kártyaolvasókat, *USB*-s memóriakártyákat. Egyszerűen csak összedugjuk az eszközöket és már csatolhatjuk is be a meghajtót.
Kissé más a helyzet, ha *USB*-re csatolt *CD*, vagy *DVD* írót szeretnénk használni. Itt is fordítsuk a rendszermagba az *USB Storage Support* modult, de emellett még szükségünk lesz az *ATA-ATAPI* eszközök beállításai között található *SCSI Emulation Support* modulra, valamint a *SCSI* eszközök között *SCSI generic* és *CDROM support* modulokra. Ezután a *CD*, *DVD* írónkat csatlakoztatva az eszközt a */dev/srX* alatt találjuk, ahol *X* egy *0*, vagy annál nagyobb szám, attól függően, hogy hány eszközt csatlakoztattunk.

Én személy szerint a külső merevlemezeket is a *SCSI* illesztésen keresztül szoktam csatolni, így a kernelben még hozzáadom a *SCSI disk support* modult is a rendszermaghoz. Ezután az *USB*-n becsatolt lemezek a */dev/sdXY* eszközön keresztül érhetőek el, ahol *X* egy *a..z* közötti karakter, *Y* egy *1..n* szám, attól függően, hogy hányadik lemez, hányadik partíciójáról beszélünk.
Annyit még érdemes tudni az *USB* eszközökről, hogy mint az a csatoló nevében is benne van, ezek sorosan kötött eszközök. Ennek a ténynek komoly jelentősége lehet, ha több olyan eszközt kapcsolunk össze, amelyek nagy sávszélességet igényelnek, hiszen ilyenkor rossz esetben a sebesség drasztikusan visszaeshet. Ez pedig egy videókamera, vagy egy DVD-író esetében nem éppen jó dolog.
Ennek kiküszöbölésére érdemes ilyen eszközök esetén *USB 2.0*-s csatolót használni, ami az 1.1-es 12 Mbit/sec sebességéhez képest több 400 Mbit/sec-os átviteli sebességre képes. Ilyen átviteli sebesség mellett már nyugodtan használhatunk olyan sávszélességzabáló eszközöket is, mint a már említett DVD-író, külső merevlemez, vagy sokcsatornás hangkártyák.

FireWire, a tüzes vezeték

A másik olyan eszköz, amelyet kifejezetten olyan eszközökhöz terveztek, amelyekhez nélkülözhetetlen a nagy sebességű kapcsolat, az akár 400 Mbit/sec sebességre képes *FireWire*, vagy *IEEE1394*-es szabványú csatoló.
A legújabb notebookok nagyobb hányadán már található *FireWire* csatoló, amelyen keresztül különböző nagy sebességű eszközöket – például videókamerákat, merevlemezeket – köthetünk a számítógépünkhöz. (Itt jegyezném meg, hogy a *FireWire* csatolók használhatóak hálózati eszközként is, de erről majd talán egy későbbi cikkben lesz szó.)
Természetesen a *FireWire* eszközök csatolásának első lépése itt is a megfelelő modulok befoglalása a rendszermagba. Az *IEEE 1394 (FireWire) support* menüpont alatt található modulok közül a videókamerák életrekeltéséhez az alábbi modulokat szoktam használni: *OHCI-1394 support*, *OHCI-1394 Video support*, *OHCI-DV I/O support*, *Raw IEEE1394 I/O support*. Amennyiben merevlemezt szeretnénk *FireWire*-n keresztül elérni, akkor szükségünk lesz a *SBP-2 support (Harddisks etc.)* modulra, valamint a *DMA* kihasználásához a *Enable Phys DMA support for SBP2* modul bejelölésére.
Ha elkészültek a modulok, vagy újraindultunk a friss rendszermaggal, akkor telepítsük fel a *gscanbus* csomagot, amely egy az `usbview`-hoz hasonló funkciójú programot tartalmaz. A `gscanbus` indítása után szintén grafikus felületen rögtön megkapjuk a gépünkhöz csatolt *FireWire* eszközök topológiáját, valamint az egyes eszközökre kattintva bővebb információkat kaphatunk az adott eszközről.
A videókamerám esetében például arra is lehetőség van, hogy a számítógépről vezéreljem annak működését.
Ezzel cikkem végére is értünk volna. Úgy gondolom, most is sok hasznos eszközzel és beállítással ismerkedtünk meg. Mindenkit arra buzdítanék, hogy játszadozzon el a közelében fellelhető ilyen eszközökkel, próbálgassa a beállításokat, ugyanis sok érdekességet tartogat számunkra a *Linux*.

Illés Viktor

A felhasználói viselkedés vizsgálata (2. rész)

A sorozat előző részében megismerkedtünk a digitális Nagy Testvér módszereinek elméleti lehetőségeivel. Tudjuk tehát, hogy mit és miért lehet és érdemes megvalósítani. Most lássuk hogy hogyan!

Adatgyűjtés általában

Az internet, azon belül is a weblapok hálózata folyamatosan, és megdöbbentő mértékben növekszik, akár a forgalmat, akár a weboldalak összetettségét és számát vizsgáljuk. Ezzel a folytonos növekedéssel természetesen a weboldalakon látható grafikának, a webkiszolgálók tervezésének, kialakításának, méretezhetőségének is lépést kell tartania. Nem kivétel ez alól a weboldalakon belüli navigáció összetettsége sem, éppen ezért a webes fejlesztési folyamatok egyik legfontosabb „bemenő paramétere" a weboldal használatának jellemzése.

A weben elérhető információ mennyisége és szerkezeti összetettsége már régóta tökéletesen lehetetlenné teszi, hogy a hely pontos ismerete nélkül, egyszerűen csak megtaláljuk azt, amit éppen keresünk. Ezt felismerve kezdték el fejleszteni az adatbányászat (data mining) mintájára a *„web mining"* módszereket. A ***web mining*** szolgál az interneten fellelhető hasznos információ felfedezésére, elemzésére és kategorizálására. A web miningnak nevezett tevékenység tulajdonképpen maga is három fő részterületre osztható:

- ***Tartalom (Content Data)***: a felhasználónak jelentéssel bíró adatok elemzése. A ***„Web Content Mining"*** az a folyamat, amely során a webes dokumentumokból „kiemeljük" a tényleges tudást.
- ***Szerkezet (Structure Data)***: azoknak a metaadatoknak az összessége, amelyek meghatározzák egy szervezet webes információs rendszerének logikai struktúráját. A ***„Web Structure Mining"*** során a tudást (információt) tehát az adott honlap szerkezete hordozza.
- ***Használat (Usage Data)***: olyan adat, amely a felhasználók webes interakciói során keletkezik. A ***„Web Usage Mining"*** célja az, hogy szabályosságokat fedezzen fel a felhasználók viselkedésében, lépéseiben, vagy például abban, hogy mit töltenek le.

A szakirodalom a felhasználói viselkedés vizsgálatát „Web Usage Mining", míg a modellezéshez szükséges adatok előállítását „Usage Mining" néven említi.

Amint azt az előző részben már említette, a viselkedés modellezése alapvetően két részből áll: az adatgyűjtésből és az adatok elemzéséből. A gyűjtés feladata olyan adatok előállítása, amelyek megfelelően részletesek ahhoz, hogy az analízist végző szoftverek használható modelleket alkothassanak belőlük. Mivel nem célunk az, hogy minden egyes látogató viselkedését egyenként elemezzük, csupán az „átlagos látogató" paramétereire vagyunk kíváncsiak, ezért a megszemélyesítést lehetővé tevő adatokra (tipikusan IP cím és felhasználónév) csak a felhasználói munkamenetek elkülönítéséhez van szükségünk.

Amennyiben a felhasználókat más módszerrel is meg lehet különböztetni (például rendelkeznek egyedi munkamenet azonosítóval), akkor semmilyen személyes adatra nem lesz szükségünk, hiszen elegendő az oldal neve és megtekintésének időbélyege.

Az adatgyűjtési módszerek alapvetően három csoportra oszthatóak. A megfigyelést végző kódot beépíthetjük magába az oldalba, illetve végezhetünk adatgyűjtést az ügyfélen és a kiszolgálón. Mint megannyi más területen, itt is mindegyik módszernek megvannak a maga előnyei és hátrányai, amelyekkel tisztában kell lennünk, mielőtt alkalmazni kezdjük őket. Lássuk tehát!

Weboldalba épített adatgyűjtés

A módszer lényege, hogy már a portál tervezésekor kiemelt figyelmet szentelünk annak, hogy a felhasználók tevékenységéről megfelelő információval rendelkezzünk. Gyakori megoldás, hogy a weboldal elérése regisztrációhoz kötött, és csak a regisztrált és beléptetett felhasználók érhetik el a hasznos információt hordozó belső oldalakat.

Ez a módszer komoly fejlesztési ráfordítást igényel, hiszen már a tervezést is számos ponton befolyásolja.

Ha nem a kellő körültekintéssel járunk el, az később hátrányosan befolyásolhatja a rendszer fejleszthetőségét, vagyis éppen azt, amiért az adatokat gyűjtjük, és a rendszer használatát modellezzük. Hiába tudjuk az adatok alapján, hogy milyen szerkezeti változásokat kellene végrehajtanunk, ha a gondatlan tervezés miatt nem tudjuk azokat elvégezni.

Az adatgyűjtést végző programkódok alig, vagy egyáltalán nem újrahasznosíthatóak, hiszen minden egyes weboldal felépítése – ha olykor igen csekély mértékben is de – eltérő. Ugyanakkor éppen azért, mert a kódot maga az oldal tartalmazza, egészen pontos információkat kapunk külön-külön minden egyes felhasználó viselkedéséről. Ez még akkor is

igaz, ha az illető a regisztráció alkalmával valótlan adatokat adott meg. Ráadásul megfelelő kialakítás esetén ez a módszer képes rögtön előszűrt adatokkal ellátni az analizálást később elvégző alrendszert.
A regisztráció és belépési kötelezettség azonban egyrészt elriaszthatja az anonimitásukat megőrizni igyekvő felhasználókat. A legtöbb ember a magánszférába való túlzott behatolásként értékeli az ilyen jellegű működést, illetve zavarja az a tudat, hogy egy lelketlen rendszer minden lépését figyeli. Megjegyzendő persze, hogy a portál megfelelő tartalommal ellensúlyozhatja a magánszférába való behatolást. A magánszféra megsértésének problémáját ki lehet küszöbölni úgy is, ha nem az egyes felhasználók műveleteinek sorozatát mentjük, hanem csak azt, hogy a felhasználóink mit csináltak a portálon belül. Ilyenkor az adatokat olyan formában tároljuk, hogy azokból eleve lehetetlen legyen megmondani mondjuk azt, hogy személy szerint Teszt Tamás merre barangolt az oldalak között. Természetesen ehhez az egészhez az is fontos, hogy a látogatók valamelyest megbízzanak a portált működtető szervezetben, hiszen a programkódokba nem lát bele az egyszerű felhasználó. Nyílt forrású szoftverrel ez a probléma részben feloldható, de az adatvédelmi elvek betartásának ellenőrzése komoly programozási ismereteket igényelhet.
Mint említettem, ennek a megoldásnak az is előnye lehet, hogy eleve szűrt adatokat szolgáltathat. Pontosan tudjuk naplózni, hogy mely weboldalt mikor kérték le, az eseménynapló közvetlenül feldolgozható és a munkamenetek elkülönítésével sem kell foglalkoznunk, mert a belépéskor minden egyes felhasználónak létrejön az egyedi munkamenet azonosítója. Mivel a weboldalt előállító program a kiszolgálón fut, a módszer rendelkezik a szerver oldali adatgyűjtés összes ismert hátrányával is.

Adatgyűjtés az ügyfélen

Az ügyfél oldalán történő adatgyűjtés megpróbálja (bizonyos tekintetben szinte teljes sikerrel) a HTTP protokoll és a web felépítéséből, működéséből adódó hátrányokat kiküszöbölni.
Az alapvető módszer az, hogy az ügyfélen egy a böngészőtől független program (esetleg annak egy kiegészítése, vagyis egy „plug-in”) folyamatosan figyeli a felhasználó által végzett műveleteket: melyik weboldalt mikor kérte le, mennyi idő alatt érkezett meg az a kiszolgálóról, sőt akár az is mérhető, hogy az adott oldalt tartalmazó ablak mennyi ideig volt aktív.
Ez a bizonyos kiegészítő program többféleképpen kerülhet az ügyfélhez. Lehetőség van arra, hogy a honlap letöltésekor automatikusan betöltődjön és elinduljon a kliens böngészőjében úgynevezett beágyazott HTML objektumként. (Ilyen például egy JAVA applet, egy Flash, vagy egy ActiveX komponens.) Biztonsági okokból ez a megoldás csak korlátozottan használható. Az így letöltött program csak az adott (ahonnan letöltötték) kiszolgálóval tud kommunikálni, és nem képes a helyi géppel adatokat cserélni, hiszen csak ideiglenes fájlokat tud létrehozni, amelyek a portál elhagyásakor, de legkésőbb a böngésző bezárásakor megsemmisülnek. További probléma, hogy csak annak a portálnak a tartalmával kapcsolatban működőképes, ahonnan letöltötte a felhasználó. Utóbbi probléma részben kiküszöbölhető azzal, ha minden weboldalról letöltődik a program és egy közös helyen (például egy sütiben) tárolják a munkamenet azonosítót. További, nem elhanyagolható hátrány, hogy a sütikhez hasonlóan ez a lehetőség is letiltható a böngészőben.
Egy másik lehetőség az, ha egy a böngészőtől független, harmadik szoftver segítségével figyeljük a felhasználó webes műveleteit. Ekkor lehet a legpontosabb adatokat összegyűjteni, hiszen ez a harmadik, kiegészítő szoftver képes a böngészés előtt, alatt és után is rögzíteni a történéseket. Pontos adatokat kaphatunk a hálózati átviteli időkről, az egyes weboldalak olvasásának idejéről, megfigyelhető és mérhető a lokális és távoli átmeneti tárak (proxy kiszolgálók) hatása. Előny, hogy az így keletkező adatokat a későbbiek során nem kell előfeldolgozni sem.
Mindkét esetben szükséges, hogy a harmadik, kiegészítő program teljesítményigénye minimális legyen, hiszen ellenkező esetben a felhasználó hamarabb elhagyja a portált mert az lassú, illetve törli a programot a számítógépéről.
A módszer legnagyobb hátránya, hogy felhasználói aktivitást követel meg: magának a felhasználónak kell telepítenie és beállítania a szoftvert. További nehézség, hogy a felhasználók többségének nem fűződik érdeke az ilyen jellegű programok használatához, sőt egyre jellemzőbb a kifejezett elzárkózás.
Az ilyen programok általában nem szabványos protokollon és portokon keresztül juttatják vissza az összegyűjtött adatokat a kiértékelést végző rendszerhez, ezért könnyen előfordulhat, hogy mind az egyéni, mind a vállalati szférában egyre gyakoribb proxy és tűzfal megoldások nem engedik át az ilyen jellegű adatfolyamokat. Jellemző továbbá, hogy elsősorban vállalati környezetben a felhasználók jogosultságai igen korlátozottak, ezért nem tudnak semmilyen szoftver telepíteni.

Adatgyűjtés a kiszolgálón

A kiszolgálón történő adatgyűjtés nem igényli az ügyfél együttműködését, így nincs szükség sem harmadik fél által írt programokra, sem a felhasználó hozzáértésére és hozzájárulására. A felhasználó megkímélhető a regisztrációtól is, mert az eseménynapló a portáltól független módon keletkezik.
A szerver oldali adatgyűjtés tulajdonképpen nem más, mint az általában amúgy is működő naplózó szolgáltatás (hibakeresés, teljesítmény analízis, behatolási kísérletek felderítése) kiterjesztése a tárolt speciális paraméterekkel (böngésző típusa, előző oldal (referer), munkamenet azonosító). Az így keletkező adatsor nem alkalmas arra, hogy valós időben szolgáljon adatokkal a modellezéshez, hiszen jelentős „tisztítást” igényel éppen a szerteágazó adattartalom miatt.
A módszer hátrányai a HTTP protokoll és a világháló működéséből adódnak.
A felhasználók minél gyorsabban szeretnék letölteni a tartalmat, a vállalatok viszont igyekeznek spórolni a kommunikációs költségekkel, illetve egyes tartalmakat szűrni akarnak. (Munkaidőben például ne magánjellegű információk keresésével töltse az idejét a munkavállaló.), Éppen ezért használnak annyian átmeneti tárakat és különböző proxy technikákat.
A modern böngészőprogramok azzal is segítik a felhasználót, hogy az egyszer már letöltött weboldalakat tárolják. Előfordulhat akár az is, hogy a felhasználó a weboldalba

épített navigációs elemeket használja (linkekre kattint) és nem a böngésző „vissza” és „előre” gombjait, az eseménynaplóban mégsem jelenik meg a bejegyzés, mert a böngésző szoftvere észleli, hogy letöltött honlapról van szó és a lokális átmeneti tárból szolgálja ki a kérést.
Az ilyen „zajok” kiszűrése nem lehetséges pusztán az eseménynapló feldolgozásával. A zaj jellege és mérete nem becsülhető, hiszen az ügyfélen végzett adatgyűjtéstől eltérően itt nem áll rendelkezésre pontos adat a tényleges lekérésekről. A probléma ugyanakkor részben áthidalható a webszerver beállításával. A HTTP protokoll lehetőséget ad arra, hogy az átküldött fejlécben jelezzük: nem akarjuk, hogy a kérdéses weboldal a lokális tárba kerüljön. Ilyenkor a böngésző ugyan minden alkalommal újra lekéri azt, ugyanakkor elvesznek a lokális és távoli átmeneti tárak, illetve proxy kiszolgálók által nyújtott előnyök.
A fentiek miatt a kérések egy meghatározhatatlan része el sem jut a webkiszolgálóig, jelentősen rontva ezzel a módszer pontosságát. Ugyanakkor az átmeneti tárakban leggyakrabban a képek tárolódnak, amelyek a viselkedés modellezés során nem (feltétlenül) jelentenek értékes információt. Mindent egybevetve a legjobb megoldás talán a három módszer valamiféle összeházasítása lehetne.
A kiszolgálón történő adatgyűjtés egy másik jellegzetes problémája a nem determinisztikus hálózati működés. Ez azt jelenti, hogy egy az eseménynaplóba bekerült felhasználói kérés jelenléte nem jelenti teljes biztonsággal azt, hogy a felhasználóhoz el is jutott a kért tartalom. Akár az is lehetséges, hogy a letöltés befejeződése előtt a felhasználó bezárta a böngészőt, hiszen a webkiszolgáló erről nem kap visszajelzést. Tekintettel a mai hálózatok viszonylagos megbízhatóságára az így keletkező zaj csak kis jelentőséggel bír.
Az eseménynaplóba minden a kiszolgálóhoz érkező kérés bekerül: minden lekért dokumentumnak, képnek, videónak, zenének, animációnak, appletnek, vagy fájlnak nyoma van. Ez azt jelenti, hogy egy-egy weboldal letöltésével akár több tíz bejegyzés is keletkezhet. Ugyanakkor a modellezés szempontjából lényeges információt csak bizonyos elemekkel kapcsolatos bejegyzések hordoznak. Ezért az eseménynaplót a portál szerkezetének (vagy a kívánt modellnek) megfelelő előfeldolgozás során „meg kell szűrni”.
A modern portálfejlesztés bevált módszere, hogy a különböző weboldalakat egyetlen fájl készíti el paraméterektől (esetleg a munkamenetben tárolt adatoktól) függően. Ekkor bár a felhasználó folyamatosan egy bizonyos weblapot kér le, a tartalom mindig más. A modellezés során ugyanakkor nyilván el kell különíteni ezeket a lekéréseket, hiszen tartalmilag különböző weboldalaknak tekintendők. Az előfeldolgozás során az eseménynapló lekérések mezőjének vizsgálatával lehet elkülöníteni a formailag azonos fájlhoz tartozó, de tartalmilag különböző lekéréseket.
A POST metódussal átadott és a munkamenetben tárolt paraméterek esetén újabb problémával szembesülünk. Az ilyen lekérések sajnos nem kerülnek be az eseménynaplóba és általánosan használható módszerek sincsenek a mentésükre. Nem is szokás tárolni ezeket a kéréseket, hiszen egy-egy ilyen kérés mérete akár több megabájt is lehet, ami egy komoly forgalmú portálnál a tartalomnál akár négy-nyolc nagyságrenddel nagyobb méretű eseménynaplót eredményezne.

Mindez ugyanakkor igazi problémát csak akkor jelent, ha a POST illetve munkamenet adatok ugyanarra az oldalra kerülnek vissza rendszeresen, mint ahonnan elindították a kérést. Egyre több szakember van azon a véleményen, hogy egy URI-nak könnyen olvashatónak és érthetőnek kell lennie, vagyis legalább körülbelül ki kell derülnie belőle, hogy mi található az adott weboldalon.
Régóta biztosítják a webszerverek a kérések átírásának lehetőségét, amellyel így részben elfedhetők a POST és GET metódus miatti azonos lekérések.

Ha pédául a

```
http://www.ceg.hu/index.php?m=0&sub=1&c_id=2&r_id=3
➥&act=5
```

forma helyett a

```
http://www.ceg.hu/szolgaltatasok/hardverfejlesztes/
➥ajanlat
```

formát használjuk, akkor az átalakítást a webszerver végzi a megfelelő beállítások szerint. Az ilyen úgynevezett ***„barátságos linkek”*** nagyban megkönnyítik a felhasználói viselkedés modellezését, mert az eseménynaplóban egyértelműen és könnyen szétválogatható módon jelennek meg a vizsgálandó elemek.
Az előfeldolgozás sajnos jelentős többletterhelést jelent a modellkészítés során, így ez a módszer nem alkalmas arra, hogy online, azaz folyamatosan frissülő statisztikákat és modelleket szolgáltasson. Tipikus megoldás, hogy a feldolgozás és modellalkotás a portál legkevésbé terhelt időszakában napi rendszerességgel zajlik (általában éjszaka).
A modellezés során a legnehezebb feladat az egyes felhasználók elkülönítése. Az eseménynapló alapesetben nem tartalmaz ehhez elégséges információt, mert a proxy és tűzfal mögül érkező látogatók adatai szinte teljesen megegyeznek. Egyes webszerverekhez már létezik olyan kiegészítő modul, amely munkamenet azonosítót rendel minden felhasználóhoz. Használható ezen kívül idő alapú elkülönítés, user agent vizsgálat, illetve a referer és lekérések közötti folyamatosság vizsgálata a portál szerkezetének ismeretében.
A módszer előnye, hogy sem hardveres sem szoftveres együttműködést nem igényel az ügyféltől. Csupán a sütik támogatása szükséges a munkamenet azonosító tárolásához, nem kell azonban sem az aktív felhasználói közreműködés, sem harmadik program vagy regisztráció.
Adatvédelmi szempontból is kedvezőbb megoldás a már ismertetetteknél, ugyanis a felhasználó nem kerül olyan mértékben megszemélyesítésre, ami az a magánéletét túlzottan sértené. A ma jellemző tűzfal, proxy és dinamikus IP cím kiosztási technikák tovább gyengítik a megszemélyesíthetőséget. Az eljárás egyedüli, ám igen lényeges hátránya, hogy a felhasználó tudta és beleegyezése nélkül is keletkeznek a tevékenységével kapcsolatos adatok.
Következő cikkemben részletesen ismertetem a szükséges Apache modulokat, illetve azok beállításait.

Beszédes Balázs

FreeBSD – a szomszéd vár (2. rész)

Egy FreeBSD rendszer adminisztrációja és finomhangolása.

Azt hiszem egyet tudunk érteni, hogy két UNIX rendszer használata között nincs lényeges különbség. Azonos programokat tudunk azonos módon használni: lényeges különbségek leginkább a rendszergazda szemszögéből jelentkeznek.

Rendszerindítás

A *FreeBSD* kicsit másképp indul mint egy *Linux* rendszer, ugyanis csak egy egyszerűbb rendszerbetöltő modullal rendelkezik, amelynek alapvetően nem feladata más operációs rendszerek elindítása. Ez annyit jelent, hogy csak a *FreeBSD*-t, illetve a *DOS*-t és a hozzá hasonlóan induló a *Windows* operációs rendszereket képes elindítani. Ezeknél a rendszereknél ugyanis egy úgynevezett boot szektort kell beolvasni, a rendszer elindítását már ez a kis program végzi. A jelenkori *Linux* rendszerek viszont ennél már egy kicsit összetettebb indítást igényelnek, lévén pontosan meg kell határozni a gyökér-fájlrendszer elérését, esetleg egy *initrd* elindítása is szükséges lehet. Ez utóbbi feladatokat a *FreeBSD* rendszerindítója nem képes felvállalni. Ha megfordítjuk a helyzetet, akkor egy megfelelően telepített *LILO* vagy *GRUB* képes a *FreeBSD*-t elindítani: érdemes ezt a feladatot egy *Linux* rendszerre bízni.

A nulladik fázis – MBR program

Ha telepítettük a *BootMgr* programot, akkor a gépünket bekapcsolva a következő kép fogad bennünket:

```
F1 FreeBSD
F2 DOS
F4 ??

Default: F1
```

A *BootMgr* az adott partíció szerkezetének megfelelő szöveget írja ki, ahol rendre a merevlemez partícióit láthatjuk. Amelyik partíció nem aktív, de létezik, ott kettő kérdőjelet láthatunk név helyett.
A logikai partíciók nem látszanak, helyettük csak a kiterjesztett partíció látható (szintén kettő kérdőjellel). Az alapértelmezett mindig az utoljára használt operációs rendszer lesz, bizonyos idő elteltével ez el is fog indulni. A kívánt operációs rendszert a megnevezett funkcióbillentyűvel tudjuk kiválasztani. A *BootMgr* program másolata a */boot/boot0* állományban található.

Az első fázis – Boot program

A *BootMgr* (vagy egyéb hasonló) program betölti a partíció elején található úgynevezett boot szektort, amely a FreeBSD indításának első fázisát tartalmazza. Sok tevékenységet nem lehet belesűríteni, hiszen ennek a szektornak a mérete csak 512 Bájt (*/boot/boot1* fájl másolata); egyedüli dolga a beledrótozott címről a második fázis programját beolvasni.

Második fázis – Előtöltő program

Az előtöltő program (*BTX loader*) veszi át a stafétabotot, amely közel 8 kilobájt méretű (*/boot/boot2* állomány).
Ez már elegendő méret arra, hogy a *FreeBSD* által használt *UFS* (illetve *UFS2*) fájlrendszert alapszinten kezelni tudja. Ki tudjuk választani a betöltendő rendszermagot és a betöltő (*loader*) programot. Ezek általában a */boot/kernel* és a */boot/loader* állományban találhatók. Az előtöltő beolvassa a betöltő programot, majd elindítja azt.

Harmadik fázis – Betöltő program

A */boot/loader* felel a rendszermag és a hozzá tartozó modulok betöltéséért. Ha saját rendszert fordítunk, ez a betöltő program együtt készül el az új rendszermaggal. Tartalmaz egy egyszerű parancsfelületet, ahol a főbb paramétereket módosítani tudjuk. A betöltő program feldolgoz néhány szöveges állományt is, amelyekkel a betöltést befolyásolni tudjuk. Ilyenek a */boot/loader.rc*, a */boot/defaults/loader.conf* és a */boot/loader.conf*. Ezekből a */boot/loader.conf* állományt érdemes megnézni, ahol a betöltendő rendszermagmodulokat tudjuk megadni:

```
ng_ubt_load="YES"
linux_load="YES"
nvidia_load="YES"
```

A rendszermag

Az operációs rendszerek lelke az a mag, amely feladata tömören a gépben található hardverek, erőforrások és perifériák kezelése, illetve a programok biztonságos futtatása.
A *FreeBSD* rendszermag fényes fehér szöveget ír ki a megtalált eszközökről, illetve azok nevéről; s ez a szöveg akár két-három képernyőnyi is lehet.

A rendszermag beállításai

A telepítéssel foglalkozó részben kihagytam a rendszermag beállításait, mivel egy általánosságban munkaállomásnak

használt telepítést nem befolyásol érdemlegesen. Ebben a részben viszont témába vág, s szeretném itt megemlíteni ezt a fontos műveletet.
Legegyszerűbb (de nem egyedüli!) módja ennek a beállításnak, hogy a feltelepített rendszert újratelepítjük (zárójelben megjegyzem, hogy a FreeBSD működését jobban elsajátítva három-öt hét múlva egy újratelepítéssel sok előnyhöz juthatunk: másképp nézünk a fájlrendszer szerkezetére; esetleg más programokat használunk, mint először gondoltuk; a feltett komponenseket közül esetleg mást választunk ki).
A rendszermag beállításának lényege, hogy feloldjuk az ütközéseket az egymással konfliktusba kerülő eszközmeghajtók között. Ehhez a tevékenységhez ***nagyon*** ismerni kell a gépünkben található hardvereszközöket és ezek pontos működési paramétereit (***DMA***, ***IO*** és ***IRQ*** számok). A rendszermag ugyan jó „hatásfokkal" képes a gépünkben található eszközöket helyesen beállítani, viszont előfordulhatnak olyan hibák, amelyekhez kevés lesz a tudása. Például a rendszermag olyan eszközöket is találni vélhet, amelyek nincsenek; olyan meghajtókat esetleg nem tudhat beállítani, amelyek által kezelt hardverek viszont vannak a gépünkben. Ekkor le kell tiltanunk a nem használandó eszközmeghajtókat, majd engedélyeznünk kell azokat, amelyeket ellenben használnánk. Egyre kevésbé előforduló probléma a ***PnP*** tudást mellőző ***(E)ISA*** kártyák beállítása, amelyeknél pontosan meg kell adnunk a rendszermagnak a kártya által használt paramétereket.
A rendszermag betöltésekor a betöltő program beolvassa a ***/boot/device.hints*** állományt, amely a fentieket tartalmazza szöveges formában:

```
hint.fdc.0.at="isa"
hint.fdc.0.port="0x3F0"
hint.fdc.0.irq="6"
hint.fdc.0.drq="2"
```

Ennek az állománynak a szerkezete módosulhat a rendszermag fordításakor, lévén ha az egyes eszközmeghajtókat kihagyjuk vagy beletesszük az új rendszermagba, akkor értelemszerűen ezekhez tartozó `device.hints` sorok megszűnnek vagy újak keletkeznek.
A rendszermag betöltése után a ***/sbin/init*** program veszi át a vezérlést, amely elvégzi az operációs rendszer alapvető működéséhez szükséges programok betöltését és kezelését. Ekkor a kiírt a szöveg színe már világosszürke lesz.

Az alaprendszer és a telepített programok beállításai

A legtöbb beállítási igényünket a ***/etc/rc.conf*** állományban tehetjük a ***FreeBSD*** számára nyilvánvalóvá. Üres vagy majdnem üres ***/etc/rc.conf*** állomány esetén sem kell aggódnunk, a legtöbb (alapértelmezett) beállítást a ***/etc/defaults/rc.conf*** fájlban megtaláljuk, amelyeket a ***/etc/rc.conf*** beállításai értelemszerűen felülbírálnak. A `sysinstall` program szintén ebbe az állományba írja az általunk választott beállításokat, azonban ezek a beállítások csak egy újraindítás után lépnek életbe. Természetesen nem kell minden egyes változtatás után újraindítani az operációs rendszert, de különösebb tudás nélkül ez a leggyorsabb mód.
Sok egyéb lehetőségünk is van a ***/etc/*** könyvtárban, ha az alaprendszer különféle komponenseit be szeretnénk állítani (***ISDN***, ***Bluetooth***, ***PPP***, stb). Fontos azonban tudatában lenni annak, hogy a telepített programok nem az alaprendszerrel közös könyvtárakba kerülnek, hanem a ***/usr/local/*** könyvtár alá, ahol elegendő program telepítése után határozottan felismerhető lesz egy teljes gyökérkönyvtár. Például egy feltelepített ***Apache*** webkiszolgáló program beállítása a ***Linux*** esetén megszokott ***/etc/apache/httpd.conf*** helyett a ***/usr/local/etc/apache/httpd.conf*** állományt használhatjuk. Ez eleinte kicsit szétszórtnak tűnik, de hamar megtanulható, hogy az alaprendszeren kívüli összes program a ***/usr/local/*** alatt található meg. Különösen akkor kettős ez az állapot, ha az alaprendszer szerves tartozékát külső programként is feltelepítjük (***GNU C*** fordító, ***PERL***, ***SSH***, ***Sendmail***). Segítséget jelenthet, hogy ez a „filozófia" végigkövethető a legtöbb komponens esetén. Például az alaprendszer programjait a ***/etc/rc.d/*** könyvtár alá tett kis programocskák indítják, a pluszban feltett programokat pedig a ***/usr/local/etc/rc.d/*** alatti programocskák. Például az SSH futásának ellenőrzése a ***/etc/rc.d/sshd status*** futtatásával leszünk képesek, mivel az SSH az alaprendszer része; ellenben az Apache2 elindítása a ***/usr/local/etc/rc.d/apache2.sh start*** paranccsal történhet (mivel ez nem az alaprendszer része).

A sysctl felület

A `sysctl` egy egyszerű program, amellyel a rendszermag futás közbeni beállítását ejthetjük meg. Használata teljesen azonos a Linux alatt használatos `sysctl` programmal. Ha néhány paramétert minden indítás után be szeretnénk állítani, akkor ezt a ***/etc/sysctl.conf*** állományban tehetjük meg:

```
security.bsd.see_other_uids=0
```

A fájlrendszer

A ***FreeBSD*** fájlrendszere az ***UFS (UFS2)*** nevet viseli, mint ***Unix File System*** (***UNIX*** fájlrendszer). Némileg eltér a ***Linux*** alapvető (***Ext2***) fájlrendszerétől, de alapelveiben azonosak. Egy új fogalmat azonban ismerni kell hozzá: ez a ***Soft Updates***. Ennek megértéséhez szükséges ismerni a fájlrendszerek néhány tulajdonságát.
A fájlrendszerek által tárolt adatok alapvetően két részre oszthatók: adatokra és metaadatokra. Az adatokkal mindenki találkozott már, hiszen ezekkel dolgozunk. A metaadatok ezekről a tárolt adatokról szolgáltatnak információkat a felhasználók számára is, de főleg az operációs rendszer felé.
A számítógépes „ókorban" minden adatot szinkron írtak a tároló eszközökre, vagyis a másolás azonnal megtörtént, a programok megvárták a kiírás végét. Ez ***PC*** esetén egészen a ***DOS*** operációs rendszer néhány utolsó verziójáig is így történt. Fel sem merült akkoriban, hogy a drága és kevés operatív tárat adatok átmeneti tárolására használják fel. Idővel azonban a memória olcsóbb lett, ezért kifizetődőbb volt az adatok egy részét a sokkal gyorsabb memóriában tartani, mint azonnal kiírni a lemezekre. Az adatok akkor kerültek ki a lemezre, amikor arra ideje volt az operációs rendszernek, illetve amikor ez a gyorsítótár megtelt.
A metaadatok általában azonnal kiíródtak a lemezre, mivel ezzel egyszerűbb lehetett a fájlrendszert kezelő modul.
A ***Soft Updates*** annyit javított ezen a megoldáson, hogy a metaadatok is gyorsítótárba kerültek, így sok apró állomány másolása vagy törlése sokkalta gyorsabb lehetett.

Hátrányaként a sérülékenyebb fájlrendszert lehet megemlíteni. Sajnos a *FreeBSD* naplózós fájlrendszert jelenleg még nem támogat rendszerszinten, azonban több éve hallani pletykákat ilyen fájlrendszer alkalmazásáról. Reménykedni tudunk csak, hogy ez a közeljövőben valósággá válik, addig is kerülő megoldásként az *5.x* sorozat már képes váratlan újraindulás (áramszünet vagy rendszerhiba) után háttérben lefuttatni a fájlrendszer ellenőrzését.

A sysinstall program

Mint már említettem, ez a program a *FreeBSD* mindenese, amely helyileg a */usr/sbin/* könyvtárban található. Használatához alapvető angol tudás szükséges, mivel magyarosítása egyelőre még tervben sem szerepel.
A telepítés során kénytelenek voltunk megismerkedni a használatával, meglepetés lehet azonban, hogy a feltelepített rendszer esetén is azonos menürendszert láthatunk, amelyből az első utáni három menüpont a telepítést indítaná el. Legyünk mindig körültekintőek a használata során, bár egy újabb telepítést nehéz akaratlanul véghezvinni, de jobb a békesség.
A telepített rendszer beállításához igazából egyetlen menüpontot használhatunk a főmenüből, s ez a *„Configure”* nevet viseli. Ebbe belelépve a leggyakoribb alaprendszert illető beállításokat meg tudjuk ejteni.

Új alaprendszer letöltése

A *FreeBSD* egyik jó tulajdonsága, hogy az alaprendszer folyamatosan változik, s ezt a változást könnyedén követhetjük a telepített rendszerünk esetén. Ehhez egy kicsit bele kell mélyedni a *CVS* működésébe, amely egy egyszerű elvet valósít meg: sok fejlesztő közös munkáját hangolja össze. Maga a *CVS* használata nem egyszerű tevékenység, viszont remek programok készültek e tevékenység egyszerűsítésére. Mivel a *FreeBSD* használói között elenyésző az alaprendszert fejlesztők száma, ezért a készítőknek egyszerű és könnyen használható eszközt kellett adni a rendszergazdák kezébe. Ez az eszköz *cvsup*, amely egyetlen feladata, hogy a *FreeBSD CVS* kiszolgálójáról (vagy annak tükörszervereiről) letöltse a legújabb alaprendszer és rendszermag forrását. Erre egy egyszerű *FTP* kapcsolat is elég lenne, viszont a teljes forrás 400-450 megabájt méretet jelent. Ezért aztán a cvsup képes arra, hogy kiderítse a különbségeket és csak azzal terhelje a hálózatot. Ezáltal alig néhány száz kilobájt forgalmat okoz heti rendszerességgel történő ellenőrzéseket alapul véve.
A legelső probléma az szokott lenni, hogy nincs feltelepítve a `cvsup` program (noha szerintem lehetne az alaprendszer része is :), ezért fel kell telepíteni. Ennek egyik módja a `sysinstall` programot használva a *„Configure”*, majd a *„Packages”* menüpontba lépve a telepítési médium kiválasztása után megkeressük és feltelepítjük a `cvsup` programot. Másik módja, hogy parancssorban kiadjuk a `pkg_add -r -v cvsup` parancsot, amely megpróbálja letölteni és feltelepíteni. Igazából a `sysinstall` is a `pkg_add` programot fogja meghívni, de mégis egyszerűbbnek látszik a használata.
A programok telepítéséről azonban a következő részben sokkal részletesebben foglalkozok. A következő probléma a `cvsup` helyes és jól működő beállítása. Ehhez egy úgynevezett `sup` állományt használunk, amely például így nézhet ki (az állomány neve lehet például *src-sup*):

```
*default host=cvsup.hu.freebsd.org
*default base=/usr
*default prefix=/usr
*default release=cvs tag=RELENG_5
*default delete use-rel-suffix
*default compress

src-all
```

Itt megadjuk a használandó kiszolgáló nevét, néhány fontos és kevésbé fontos paramétert, illetve a használandó ágat. Ez utóbbi több szót érdemel, ennek hibás beállításával hamar megszabadulhatunk a teljes */usr/src/* tartalomtól is, ugyanis nem létező tag-et használva a *cvsup* egyszerűen letörli az eddig használt forrásokat. Természetesen használhatjuk például a `RELENG_5_2` tag-et is, melynek hatására megmaradunk az *5.2* verziónál, s csak a biztonsági frissítések fognak újdonságot jelenteni. Ez utóbbit akkor érdemes használni, ha olyan programokkal dolgozunk, amelyek egy jól meghatározott *FreeBSD* verzióhoz kötődnek. Javaslom a `RELENG_5` használatát, amely esetén királyi utunk lesz az *5.x* verziók közötti váltás során, illetve ezzel biztonságosan végig tudjuk követni a stabil *5.x* ágat, minden stabilnak ítélt újdonságról értesülünk a */usr/src/UPDATING* állományt böngészve. A források letöltését a `cvsup src-sup` parancs kiadásával indíthatjuk el, a program folyamatosan információkkal szolgál az éppen módosított állományokról.
Az alaprendszer nem igényel különösebb beállítást, mivel minden ilyen irányú tevékenységet a */etc/* könyvtárban hajthatunk végre, a rendszermag könnyedén a gépünkre szabható. Ehhez meg kell keresnünk a */usr/src/sys/i386/conf/* könyvtárt, benne pedig a *GENERIC* állományt: ez tartalmazza a kernel beállításait. Szerkezete egyszerű, opciókat és eszközöket sorol fel, amelyek közül kivehetjük a szükségtelen eszközöket (például *RAID* és *SCSI* kártyák), illetve hozzávehetünk szükséges opciókat (például *QUOTA*). Célszerű a *GENERIC* állományról egy másolatot készíteni, hogy bármikor visszatérhessünk egy működő beállításhoz. Az új alaprendszer és rendszermag fordításáról illetve telepítéséről a következő részben írok.

Auth Gábor (auth.gabor@enaplo.hu)
Egy pécsi középiskolában informatikát és programozást oktat. Tíz éve botlott először a UNIX rendszerekbe, 7 év Linux használat után kapta el a FreeBSD lázat, amiből máig nem tudott kigyógyulni.

KAPCSOLÓDÓ CÍMEK

A FreeBSD projekt honlapja ➲ http://www.freebsd.org,
A magyar FreeBSD honlap ➲ http://www.freebsd.hu,
A magyar BSD honlap ➲ http://www.bsd.hu,
A kézikönyv magyar fordítása
➲ http://www.enaplo.hu/FreeBSD/handbook/.

A PLplot függvényábrázoló könyvtár

Elérkezett a C nyelven használható diagram varázslók ideje! Egy merész állítás bizonyítása következik.

Egész éjszaka számolt a gép – de az eredményt jelentő számsor önmagában nem mond túl sokat. Rengeteg munka volt az algoritmus megalkotása, viszont a látványra éhes kívülállókat a puszta számadatok nem győzik meg. Mit lehet ilyenkor tenni? Az egyik lehetséges megoldás szerint az eredménylistát bemásoljuk egy táblázatkezelőbe, majd meghívjuk azt a „tündért", vagy „varázslót" (mikor kit), aki másodpercek alatt mutatós grafikont állít elő adatainkból. Ha viszont erre a saját programunkból mi magunk is képesek vagyunk, miért választanánk a kerülőutat?

Kedves naplóm!

Valószínűségszámítást hallgatok az egyetemen. Az előadó szép számmal ad olyan feladatokat, melyek kísérletek ezerszer, vagy akár több tízezerszer történő elvégzésén és megfigyelésén alapulnak, és így számítógépes szimulációt igényelnek. Az eredményekből viszont csak úgy kaphatunk átfogó képet a megfigyelt jelenségekről, ha grafikusan is ábrázoljuk az eredményeket. Mivel nem voltam hajlandó föladni a *C* nyelv használatát, kénytelen voltam keresni egy olyan könyvtárat, amivel kedvenc programnyelvemből is készíthettem látványos bemutatót.

Igen egyszerű módszerrel álltam neki a keresésnek. Miután Debian Woody-t használok, elindítottam a `dselect` nevű csomagkezelő segédprogramot, majd leütöttem a / gombot és beírtam, hogy `plot`. Több alkalmazás nevében is szerepelt ez a szó, de csak egy olyat volt közöttük, amihez volt súgó csomag is. Azonnal telepítettem tehát a ***plplot***, ***plplot-dev***, és a ***plplot-doc*** csomagokat. Első lépésként a mellékelt példaprogramokat szerettem volna lefordítani, de erőfeszítésem kudarcba fulladt.

A *gcc* feloldhatatlan hivatkozások (undefined reference) tömkelege miatt dobott hibaüzeneteket. Ez természetes, gondoltam, hiszen nem adtam meg neki, hogy melyik függvénykönyvtárat kell hozzáfűzni (link) a programhoz. Póriasan fogalmazva nem mutattam meg, hogy melyik fájlban vannak a ***PLplot*** függvények. Vakon bízva a leírásban nekiveselkedtem a könyvtárhoz mellékelt 80 oldalas *.ps* dokumentumnak, ám ezzel kapcsolatban nem találtam semmit. Ekkor döntöttem úgy, hogy ellátogatok a ***PLplot*** hivatalos oldalára.

A ➲ *http://www.plplot.org/* oldalon ugyanazzal az állománynévvel már egy enyhén szólva kibővített, 160 oldalas leírás köszöntött, amit azon nyomban le is töltöttem. A kinyomtatott változat böngészése közben ráakadtam a megoldásra. A `plplot-config` szkript megfelelő módon történő meghívásával rövid úton letudhatjuk a fordító paraméterezését. Diadalittasan ütöttem be egyenként a szkript nevének betűit, de a gúnyosan villogó kurzor felett csak egy rideg elutasítás fogadott a ***Bash*** részéről: „`a parancs nem található`".

A megoldás küszöbén nem rettenthet el egy ilyen apróság, gondoltam, így letöltöttem a ***PLplot*** legfrissebb forrását. Az 5.3.1-es változat kitömörítése után a megszokott s

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

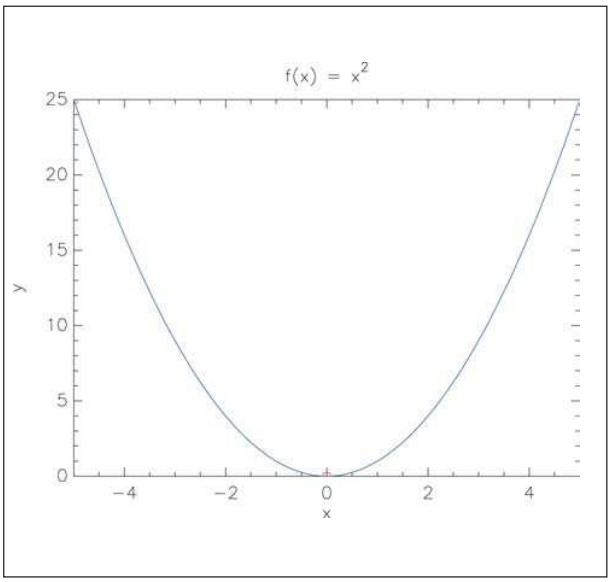
lépéseket követve kisvártatva egy működő ***PLplot*** változat büszke tulajdonosává váltam. Friss függvénykönyvtár, friss leírás és egy `plplot-config` parancsállomány fogadott a */usr/local* különböző alkönyvtáraiban. A példaprogramok már gond nélkül fordultak, így ezzel a fegyverarzenállal bátran vághattam neki a programfejlesztésnek.

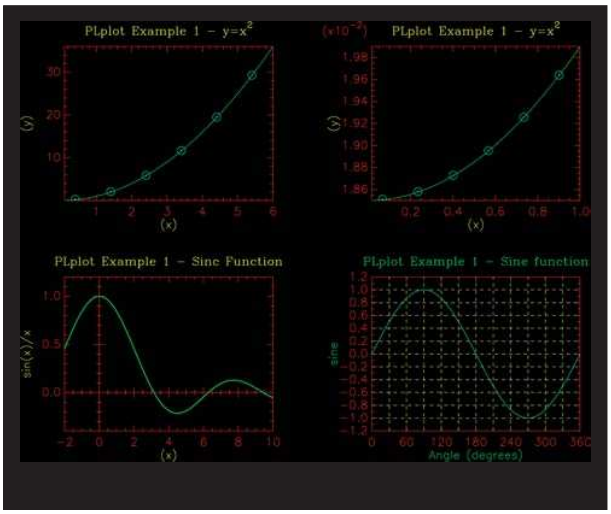
Egyfajta varázslat

Vágjunk bele a sűrűjébe! Az alábbi program egy közönséges parabolát ($y=x^2$ függvényt) ábrázol.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <plplot.h>

int main ()
{
    const PLFLT a = -5., b = 5., step = .1;
    const PLINT points = ceil ((b - a) / step) + 1;
    PLFLT *x, *y, t, min[2], max[2]; int i;

    if (NULL == (x = (PLFLT *) malloc (sizeof
      ➥(PLFLT) * points))) {
        perror ("malloc");
```

```
        return 1;
    }
    if (NULL == (y = (PLFLT *) malloc (sizeof
      ➥(PLFLT) * points))) {
        perror ("malloc");
        return 2;
    }

    min [0] = min [1] = max [0] = max [1] = 0;
    for (i = 0, t = a; t <= b; i ++, t += step) {
        x [i] = t; y [i] = pow (t, 2);
        if (min [1] > y [i]) {
            min [0] = x [i]; min [1] = y [i];
        }
        if (max [1] < y [i]) {
            max [0] = x [i]; max [1] = y [i];
        }
    }


/*****************************************************/
    plscol0 (0, 255, 255, 255);
    plscol0 (1, 0, 0, 0);
    plscol0 (15, 255, 0, 0);
    plinit ();
    plenv (a, b, min [1], max [1], 0, 0);
    pllab ("x", "y", "f(x) = x#u2#d");

    plcol0 (9);
    plline (points, x, y);

    plcol0 (15);
    plpoin (1, &min [0], &min [1], 0);
    plcol0 (15);
    plpoin (1, &max [0], &max [1], 0);

    plend ();

    free (y); free (x);
    return 0;
}
```





Mielőtt részletesen elemeznénk a program működését, próbáljuk ki! Indítás után azonnal egy kérdéssel szembesülünk:

```
Enter device number or keyword:
```

ENTER-t ütve rögtön szemünk elé tárul a gyönyörű grafikon, kék színnel jelezve magát a görbét. A legkisebb és a legnagyobb függvényérték egy-egy piros négyzettel van kiemelve. (Mint tudjuk, az x^2 függvénynek nincs maximuma, ezért őszintén remélem, hogy a matematika iránt olthatatlan szerelmet érző olvasók elnézik nekem ezt a fogalmazási pontatlanságot. Később visszatérek rá, hogy miért engedtem meg magamnak ezt az eléggé el nem ítélhető hanyagságot.)
Térjünk vissza a program által feltett kérdésre. Ez arra vonatkozik, hogy a ***PLplot*** különböző függvényein keresztül előállított ábra milyen eszköz segítségével kerüljön megjelenítésre. Az első lehetőség – amely egyben az alapértelmezett is – a grafikus felületen, vagyis egy X ablakban való megjelenítés. Ezen kívül a ***PLplot*** a legkülönfélébb eszközökhöz rendelkezik meghajtóval. Attól függően, hogy mit adtunk meg a fordítás során választhatunk ***Postscript***, ***Xfig*** és ***JPEG*** kimenet között.
Maga a program a beilleszthető fejlécállományok sorával kezdődik. A `stdlib.h` a dinamikus memóriakezelés, a `math.h` a hatványozást biztosító `pow()` függvény, míg a `plplot.h` a cikk tárgyát képező függvénykönyvtár definícióihoz szükséges. A `plplot.h` tartalmazza ezen kívül a `PLFLT` lebegőpontos, illetve a `PLINT` egész típusokat is, melyek a könyvtár használata esetén igen hatékonyan alkalmazhatók a számábrázoláshoz.
A program belépési pontja, amely jelen esetben az egyetlen függvény, a `main()`. Az első három sorban vezetjük be az új változókat. A ***PLplot*** segítségével X-Y diagramokat tudunk megjeleníteni. Az ábrázolás az a és `b` által meghatározott zárt intervallumban fog történni, az ábrázolási lépésközt pedig `step` jelzi. A `points` nevű

változóban tároljuk a ténylegesen kiszámítandó, és megjelenítendő pontok számát. Az összes többi pont számítása interpolációval történik.
Az X-Y diagram pontjai egy `x` és egy `y` nevű tömb elemei lesznek, természetesen úgy, hogy a két tömb azonos indexű tagjai alkotják egy pont koordinátáit. Bevezetünk egy `t` segédváltozót, egy-egy két elemű tömböt a minimum, és a maximum ábrázolásához, valamint egy `i` ciklusváltozót. Ez után két `malloc()` hívás következik, melyekkel területet foglalunk az `x` és az `y` tömbnek. Természetesen meg lehetett volna ezt oldani statikus tömbökkel is, ám ez rontana a program rugalmasságán.
A `min`, illetve `max` tömbök minden elemének 0 kezdőértéket adunk. Az ezt követő ciklusban számoljuk ki a görbe egyes diszkrét pontjait, illetve meghatározzuk a függvény minimumát és maximumát. Az `i` változót csupán az `x`, és `y` tömbök indexeléséhez használjuk, míg a `t` futó változó az intervallum pontjait veszi fel a megadott léptéknek megfelelően. A ciklus lényegét jelentő első sor a ciklusmagban magáért beszél. Az utána álló két elágazás egy megszokott módszer legalacsonyabb, illetve legmagasabb érték keresésére.
A megjegyzés után álló rész használja ki a ***PLplot*** függvénykönyvtár adta lehetőségeket. A ***PLplot*** biztosítja a programozót arról, hogy bármilyen eszközt is használ a megjelenítéshez, egy 16 színű paletta mindig rendelkezésére áll. Ennek a 0. színe jelenti az alapértelmezett hátteret, és az 1. az előteret. Ha ezeket nem módosítjuk, fekete háttér előtt piros színnel rajzol a könyvtár, ami a képernyőn még tűrhető, de nyomtatásra nem használható. Éppen ezért az első három sorban módosítjuk a palettát.
Nulladik színnek fehéret, elsőnek pedig feketét állítunk be (RGB megadás). Azért, hogy a piros továbbra is elérhető maradjon, beírjuk a 15. szín helyére. A `plinit()` függvényt minden ábrázolás előtt meg kell hívni, mivel bizonyos belső változóknak ez ad kezdőértéket. Lehetőségün van arra, hogy a `plinit()` meghívása előtt a programból meghatározzuk a kimeneti eszközt a `plsdev()` függvény segítségével. Ha ez elmarad, akkor a `plinit()` felkínál egy listát a felhasználónak az elérhető eszközökről, és az ő döntésén múlik, hogy az eredmény milyen formátumban jelenik meg.
A `plenv()` az ábrázolás környezetét állítja be. Első két paramétere az ***x***, második kettő az ***y*** tengelyen vett alsó és felső korlát. Az ***x*** tengely beállításához mi az a és `b` pontokat vettük, az ***y***-hoz pedig a számított minimum és maximum értékét. Többek között e paraméter miatt használtam az imént olyan pongyola módon a minimum és maximum fogalmát. A függvény ötödik paramétere adja meg, hogy a két tengely skálája azonos legyen-e (1), vagy sem (0).
Az utolsó paraméter a különböző címkék, tengelyek, rácsok megjelenítését állítja be. Mi itt csak egy dobozt kérünk a diagram köré, de az egyéb lehetőségekről bőven találunk leírást a dokumentációban.
A `pllab()` függvény segítségével határozhatunk meg címkéket az ***x*** és ***y*** tengelyhez, illetve itt adhatjuk meg a diagram címét. A karakterláncokat, amelyek különféle módosítókat is tartalmazhatnak, a jelzett sorrendben kell átadni,. Egy módosító mindig # karakterrel kezdődik, és hasonló beállításokra használható, mint a legtöbb szövegszerkesztő ***Formátum/Karakter...*** menüpontja. A `#u` az angol ***up (fel)*** jele, míg a `#d` a ***down (le)***. A bemutatott módszerrel a 2-es a felső indexbe kerül. Ha netán magára a # karakterre lenne szükségünk, azt a ## jelsorozattal jeleníthetnénk meg. Ha valamelyik címkét nem szeretnénk megadni, ott az üres karakterláncot kell átadni és nem ***NULL***-t!
Ezután jön a görbe megrajzolása. Előbb az előtérszínt állítjuk be kékre a `plcol0()` függvény segítségével. Az alapértelmezett palettában a kék a 9-es szín. Utána a `plline()` függvényt használjuk, mely két tömb megadott számú eleme alapján hoz létre egy képet. Első paraméterként átadjuk a pontok számát, második és harmadik paraméterként az `x` és `y` tömböket. Utóbbiak azonos indexű elemei alkotják egy pont koordinátáit. A függvény a szomszédos pontokat egy vonaldarabbal összeköti.
A maximum és minimum bejelölése is hasonlóan történik. Először beállítjuk pirosra az előtérszínt. Vegyük észre, hogy azért kapunk pirosat a 15-ös kiválasztásával, mert amikor az elején átrendeztük a palettát, ezt állítottuk be. A `plpoin()` hasonlít a `plline()`-hoz, de nem húzza össze a szomszédos pontokat és lehetőség van a pontok karakterének megválasztására. Első paramétere az ábrázolandó pontok száma, második és harmadik az x és y tömbök. Mivel egyetlen pontot ábrázolunk, kénytelenek voltunk a változó címét képezni, hogy hasonlítson egy tömbre. Az utolsó paraméterrel a 0-ás karaktert választjuk ki, ami épp egy négyzet.
Legvégül meghívjuk a `plend()` függvényt. Ez addig nem tér vissza, amíg a rajznak nincs vége. Ez egy ***Postscript*** dokumentum esetén a nyomtatás végét, egy ablak esetén az ablak bezárását jelenti. A `plend ()` visszatérésével a rajz megszűnik és a belső változók felszabadulnak. Mivel dinamikusan foglaltunk helyet az `x` és `y` tömböknek, ezeket is felszabadítjuk egy-egy `free()` hívással a program végén.
A `return 0;` utasítással az alkalmazás véget ér.

Hogyan tovább?

A fenti példaprogram amolyan állatorvosi ló, amin mindent ki lehet próbálni. Megpróbálkozhatunk például a ***sin(x)*** függvény ábrázolásával. Ehhez nem kell mást tennünk, mint a `for (;;)` ciklus magjában megfelelően módosítani azt sort, amely értéket ad az `y[i]` tömbelemeknek.
A ***PLplot*** természetesen sokkal többet tud annál, mint amit ebben a cikkben megmutattam. Nem csak ***C***-hez, de ***C++***-hoz, ***Fortranhoz***, ***Perlhez***, ***Tcl/Tk***-hoz, és ***Java***-hoz is van felülete, így szinte bárhol használható. Sőt, működik ***Microsoft Windows*** alatt is! A ***Visual Studio***-val könnyedén le lehet fordítani könyvtárat, akár ***DLL***-t is lehet belőle készíteni. Ez egyben azt is jelenti, hogy a ***PLplot***-tal készül programok forráskód szinten hordozhatóak. A könyvtár fejlesztése nem állt meg, folyamatosak a frissítések.
A ***PLplot*** lehetőséget biztosít oszlop-, torta-, és 3 dimenziós diagram létrehozására, vannak benne különböző vonal- és kitöltési stílusok, valamint különleges betűkészletek is. Akit érdekel a grafikus ábrázolás, mindenképpen töltse le a leírást, és szánjon rá egy-két délutánt az átolvasására. Megéri foglalkozni vele, mert egy idő után gyerekjáték a használata. Sok sikert hozzá!

Fülöp Balázs

PHP 5 – Új generáció

...avagy hogyan használjuk okosan az osztályokat, és objektumokat PHP 5-ben.

Annak idején, amikor a *PHP 3*-at leváltotta a *PHP 4*, egy hasonló átállásnak lehettünk tanúi, mint most. A programnyelv fejlesztői nagyon odafigyeltek arra, hogy az új rendszerrel kompatibilis maradjon a kód, vagyis igyekeztek minimálisra csökkenteni azoknak a programoknak a számát, amelyek nem futnak az új *PHP*-vel. Tudjuk, hogy az átállás nem mindig volt sikeres, de legalább megpróbálták... És valljuk be, azért nem olyan rossz az eredmény, amit elértek. Az új változat tartalmazott néhány új megoldást, pár koncepcionális újítást, filozófiai jellegű helyretételt, amelyek hosszú távon igen jelentősek a programnyelv teljességére, összefogottságára nézve. A nagy változás azonban az volt, hogy megjelent a ***ZEND engine 1***, ami valójában az a virtuális gép, amely a *PHP* kódot futtatja. Ezzel vált igazából platform függetlenné a *PHP*.

Hasonló változásokról beszélhetünk mostanság is, a *PHP 5* megjelenésénél. Ami a programozást illeti, itt is olyan jellegű módosítások történtek, mint annak idején a *PHP 4*-ben. Két fő változat közben a függvény könyvtárak alakulása, bővülése természetesen folyamatos, ezzel nem is szükséges foglalkoznunk. Ebből a szempontból a *PHP 4* vége és a *PHP 5* eleje gyakorlatilag ugyanazt tudja, ám a *PHP 5* bevezetéséhez időzítettek jó néhány teljesen új függvényt, csakhogy könnyebb legyen az életünk. Ami igazán új, az a már említett virtuális gép a *ZEND*, amely már a 2-es változatszámot viseli. Ellátták néhány új képességgel, átalakult a memóriakezelés, gyorsabb lett, stabilabb lett, azonban mi programozók, ebből nem sokat látunk.

Ami viszont rendkívüli módon – mondhatni teljesen – átalakult, az az objektummodell. Ezzel a PHP talán leggyengébb bástyáját sikerült igencsak megerősíteni: olyan mértékben kibővült, hogy akár komoly dolgokra is használni lehet. A régi modell csak erős jóindulattal teljesítette azokat az alapkritériumokat, amelyek egy objektumközpontú nyelvnek sajátjai. Nem volt igazi egységbe zárás, s a többalakúság is csak úgy teljesült, ha programozóként betartottuk az ehhez szükséges szabályokat – szerencsére ez nincs többé.

Nem véletlen tehát, hogy a most induló cikksorozatunk is szinte kizárólag erről fog szólni. Egyrészt azért, hogy mindenki megismerje az új szolgáltatásokat, másrészt, hogy a tisztelt olvasó kedvet kapjon az objektumok okos, átgondolt használatához, amivel jelentősen megkönnyítheti saját dolgát – ha programozni támad kedve. Éppen ezért – a sorozat alcímének megfelelően – az alapoktól kezdve, a szintaktika mellett a szemlélet áttekintésével próbálom elővezetni mondandómat. Remélem az Olvasó is méltónak találja majd írásomat az ott említett jelzőre.

Osztályok és objektumok

Mint tudjuk, az osztályok olyan kerek, egész szerkezetek, amelyekben nem csak értékeket tárolhatunk, de azok segítségével különböző műveleteket végezhetünk, utasításokat hajthatunk végre. Olyan összetett típusok tehát, amelyekben nem csak adat, hanem logika is található.

A legfontosabb változás a *PHP 4*-hez képest, amelyet mindjárt az elején meg kell említsek, hogy az osztályok példányaira, az objektumokra ezentúl referenciaként hivatkozunk. Előzőleg, ha egy objektumot átadtunk bárhogyan a programban, az lemásolódott, és egy új példány jött létre, azon végeztük azután a műveleteket. *PHP 5*-ben, ha átadunk egy objektumot, például valamilyen függvénynek, akkor annak az hivatkozásként, referenciaként adódik át, ami a gyakorlatban azt jelenti, hogy az objektum csak egyszer szerepel a memóriában, és amikor valahonnan használjuk, valójában arra a memóriaterületre hivatkozunk, ahol az elhelyezkedik. Ha tehát átmásoljuk az objektumunkat egy másik „változóba”, az igazából ugyanaz a változó marad, amit ezután két módon is elérhetünk.

Új osztály létrehozása

Új osztályt még mindig a megszokott módon kell létrehozni, azaz a `class` kulcsszóval, amit az osztály neve követ, majd kapcsos zárójelek között az osztály definíciója. Lássunk erre egy példát:

```
<?php
class Negyzet {
      private $oldal = 0;
      private $terulet = 0;

      public function oldalHossztBeallit($ertek) {
        $this->oldal=$ertek;
        $this->terulet=$erte*$ertek;
      }

      public function terulete() {
        echo $this->terulet;
```

```
        }
}
$negyzet = new Negyzet();
$negyzet->oldalHosszBeallit(7);
$negyzet->terulete();
?>
```

Ez majdnem olyan, mintha *PHP 4*-ben íródott volna. Az osztály a `new` kulcsszó segítségével példányosítható. A `$this` különleges változó arra szolgál, hogy az osztályon belül a saját tulajdonságokra, tagfüggvényekre hivatkozhassunk. Az osztály egy példánya egy konkrét elemre (esetünkben a négyzetek közül egy konkrét négyzetre) vonatkozik, ebben eddig nincs is semmi különleges.
Ha azonban jobban megnézzük a kódot, ott van az a bizonyos `public`, illetve `private` kulcsszó. Mi is ez valójában?

Nyilvánosság – az egységbe zárás kulcsa

A *PHP 5*-ben is bevezették a más objektumközpontú nyelvekben már régen használatban lévő adattulajdonságokat, amelyek arra hivatottak, hogy segítségükkel elrejtsük az osztályban szereplő tagfüggvényeket, osztályváltozókat. Erre azért van szükség, hogy az osztály által nyújtott szolgáltatásokhoz, erőforrásokhoz csak az osztályban megvalósított, szabványos felületeken keresztül férhessünk hozzá, s ne tehessünk kerülőutakat, melyek használata rövid időn belül átláthatatlanná teszi kódunkat. A fenti példa alapján képzeljük el, hogy a négyzet oldalát nem az erre írt tagfüggvényen keresztül állítjuk be, hanem közvetlenül, az $oldal osztályváltozón keresztül. Ekkor a négyzet területe természetesen nem számítódik ki, nekünk kell azt megtenni, s ha netán elfelejtenénk, az osztályunk már is helytelen működést produkál. A megoldás az, ha megtiltjuk, hogy az $oldal változóhoz közvetlenül is hozzá lehessen férni (ezt jelzi a `private` kulcsszó).
Ennek megoldására a klasszikus séma szerint három tulajdonság áll rendelkezésünkre, amelyet az osztályváltozókra és tagfüggvényekre egyaránt alkalmazhatunk.

- `public`: Ez az alapértelmezett tulajdonság. Ha nem adunk meg semmit, akkor automatikusan nyilvános lesz az osztályelem. A nyilvános elemek elérhetők az osztály példányán keresztül a -> operátor segítségével – minden korlátozás nélkül. Azokat a tagfüggvényeket tehát, amelyeket kifejezetten arra célra készítünk, hogy a program egyéb részeiből használjuk, nyilvánosnak kell deklarálni.

- `private`: Az ilyen módosítóval ellátott elemek csak az adott osztály belsejében lévő programkódból érhetők el.

- `protected`: A védett tulajdonságot majd csak később fogjuk jól megérteni. Az így jelölt elemek csak azokból az osztályokból érhetők el, amelyek az adott osztályból öröklődnek (és természetesen önmagából is). Ez valójában arra szolgál, hogy azért ne zárjuk ki teljesen az adott változót a további műveletből. Ha egy osztályt továbbfejlesztünk az öröklés módszerét választva, igen nagy korlátokba ütközhetnénk, ha nem érhetnénk el az ősosztály meghatározott változóit.

Ezeket az adattulajdonságokat helyesen alkalmazva elérhetjük, hogy az általunk írt osztályokat csak olyan módon lehessen használni, ahogyan azt mi megálmodtuk. A pontosan meghatározott használhatóság pedig lehetővé teszi számunkra, hogy építsünk rá.

Konstruktorok és destruktorok

Ha megnézzük, a fenti példában az osztály használatát, láthatjuk, hogy a példányosítás során jó volna, ha nem kellene külön beállítanunk a négyzet oldalát, feltételezhető, hogy ha egy négyzetet akarunk „csinálni”, annak van valamilyen hosszú oldala. Ilyen helyzetek megoldására találták ki a konstruktorokat. Ezek valójában különleges tagfüggvények. Automatikusan meghívódnak, amikor a `new` kulcsszó segítségével létrehozunk egy példányt, ezáltal bizonyos kezdeti értékek beállítására kiválóan alkalmasak. *PHP 4*-ben azok a metódusokat tekintette a nyelv konstruktoroknak, amelyek azonosak voltak az osztály nevével. Ez mostanra megváltozott. A konstruktorokat minden körülmények között `__consturct` névvel kell illetni. Lássunk erre egy példát, egészítsük ki a fenti osztályt az alábbi tagfüggvénnyel:

```
public function __construct($oldal) {
        $this->oldalHosszBeallit($oldal);
}
```

Most pedig példányosítsuk az osztályt úgy, hogy egyből át is adjuk az oldal hosszát:

```
$negyzet = new Negyzet(7);
```

Hasonló módon kell értelmezni a destruktorokat is, amelyek akkor hívódnak meg, amikor az osztály egy példánya megszűnik létezni. Általában arra szokták őket használni, hogy az osztály által használt erőforrásokat (adatbáziskapcsolat, fájlleíró) felszabadítsák. Destruktorok nem voltak a *PHP 4*-ben, csak most kerültek bevezetésre. Egészítsük ki a példaosztályunkat egy destruktor tagfüggvénnyel:

```
public function __destruct() {
    echo "A négyzet oldalhossza elhalálozásának
        ➥időpontjában: $this->oldal"
}
```

Ha lefuttatjuk a programunkat, akkor a program végeztével a PHP automatikusan felszabadítja a változókat, tehát minden példányra meg fog hívódni a destruktor, amely kilistázza nekünk, hogy mekkora volt a négyzet a futás végén. (Ugyanez az eredmény történik, ha kézzel, az `unset()` függvénnyel szabadítjuk fel az objektumunkat.)

Statikus változók, tagfüggvények

A statikus elemek olyan résztvevői az osztályoknak, amelyek nem élnek együtt azok példányaival, sokkal inkább kötődnek az osztályhoz, magához. Statikus elemek eléréséhez nincs is szükségünk az objektumokra, az osztályra hivatkozva érhetjük el azokat. A statikus elemeket a `static` kulcsszóval vezethetjük be, amelynek minden esetben az adattulajdonságokat jelölő kulcsszó után kell következnie.

Hogy mindezt megértsük, egészítsük ki a fenti osztályunkat az alábbi tagfüggvénnyel:

```
public static leiras() {
        echo „Én egy olyan osztály vagyok, aki
              ➥négyzeteket képvisel”;
}
```

A fenti függvénynek ugyan sok értelme nincs, de jól szemlélteti a lényeget. A statikus elemekre a `Scope (::)` operátorral hivatkozhatunk. Lássuk:

```
Negyzet::leiras();
```

Mint látható, anélkül, hogy példányosítottuk volna az osztályt, leírást kérhetünk róla. Jól látható az is, hogy az ilyen tagfüggvények nem kötődnek szorosan az osztályhoz, nem használják ki azt, hogy az osztályok adatokat és logikát tartalmaznak. Egy statikus metódus csupán „logika”, nem köthető az osztály adataihoz. Ezek után szinte természetes, hogy nem is hivatkozhatunk benne hagyományos osztályváltozókra, tehát sehol nem fordulhat elő benne a `$this` kulcsszó.
Ha már a hivatkozásoknál tartunk: megszokhattuk, hogy a `$this` változón keresztül érhetjük el többek között az osztály saját tagfüggvényeit. Ez statikus esetben nem működik, mivel ott nem is beszélhetünk példányról, ezért itt a `self` kulcsszó használata a megoldás. Ha például a konstruktorban ki szeretnénk írni, hogy mely osztályról is van szó, akkor beletehetjük a

```
self::leiras();
```

sort, amely meghozza a kívánt eredményt.
Az osztályváltozók esetén is ugyanaz a helyzet, mint a tagfüggvényeknél. Ők is lehetnek statikusak, ekkor hasonló tulajdonságok érvényesek rájuk is. Logikailag azonban némiképp bonyolítható a helyzet. Ha statikus változókat használunk, akkor nem nehéz kitalálni, hogy ezek a változók az osztály minden példányából ugyanúgy, egyediként látszanak. Ha tehát az egyik példány megváltoztatja egy statikus változó értékét, azt utána minden példány „érzékelni” fogja. Példaként valósítsunk meg a négyzeteket képviselő osztályunkban példányszámlálást, amelynek segítségével megmondható, hogy adott pillanatban hány négyzetpéldány van jelen a programban. Ehhez az alábbi módon alakítsuk át az osztályunkat:

```
<?php
class Negyzet {
    private $oldal = 0;
    private $terulet = 0;

    private static $peldanyszam = 0;

    public function __construct($oldal) {
        self::$peldanyszam++;
        $this->oldalHossztBeallit($oldal);
    }

    public static function aPeldanyokSzama() {
        echo  self::$peldanyszam;
    }

    public function oldalHossztBeallit($ertek) {
        $this->oldal=$ertek;
        $this->terulet=$ertek*$ertek;
    }

    public function terulete() {
        echo $this->terulet;
    }

    public function __destruct() {
        self::$peldanyszam–;
    }
}
$negyzet1 = new Negyzet(7);
$negyzet2 = new Negyzet(2);
$negyzet3 = new Negyzet(9);
unset($negyzet1);   //elpusztitom a negyzet1
                      ➥peldanyt
Negyzet::aPeldanyokSzama();
?>
```

A kód futáskor kiírja, hogy két darab példány van jelen épp az adott `Négyzet` osztályból. Ha ugyanis létrejön egy új, meghívódik a konstruktor, ami átállítja a számlálót, ami mint tudjuk, minden „példányt” érint. (Azért helytelen kicsit a kifejezés, mert a statikus változók a statikus tagfüggvényekhez hasonlóan igazából nem kötődnek az osztály példányaihoz – csak annyiban, hogy ha nem statikus tagfüggvényben hivatkozunk rájuk, akkor minden példány „ugyanazt” az értéket látja.)
A példában statikus tagfüggvénnyel fértünk hozzá a hasonló tulajdonságú változóhoz, mert egy „globális”, példányfüggetlen szolgáltatásra volt szükségünk, de ez nem feltétel. Nem statikus metódusokból is módosíthatjuk, lekérhetjük ezeket a változókat, sőt, ekkor válik érdekessé a dolog. (Valójában ez történt akkor is, amikor a konstruktorból, destruktorból növeltük ill. csökkentettük az osztálytulajdonság értékét, de ez nem mindig tudatosul a példa tanulmányozásakor, ezért hívtam fel rá a figyelmet.)
Nagyon fontos, hogy megértsük a statikus változók és tagfüggvények szerepét, jelentőségét, mert ezzel egy halomnyi „bűvésztrükköt” tudunk később megvalósítani – olyanokat, amelyektől tényleg „szép” lesz az, amit csinálunk.

Állandók

A ***PHP 5*** lehetővé teszi állandók (konstansok) bevezetését osztályváltozók gyanánt. Ezek nagyon közeli rokonságban állnak a fentebb emlegetett statikus változókkal. Egyik fontos különbség, hogy értékük nem változtatható meg, az egyedüli értékadás a deklaráció során történik. A bevezetés a `const` kulcsszóval történik, s fontos megjegyezni, hogy nem használjuk a `$` változóelőtagot sem a deklarációnál, sem később az érték kiolvasásánál!
A változó értékét szintén a `self` előtaggal és a `scope` operátor használatával kaphatjuk meg, s hasonlóan a statikus testvéreikhez, ezek sem érhetők el az objektumpéldányokból, csakis az osztályon keresztül.

```
<?php
class ProbaOsztaly {
        public const allando = 'barmi';

        public function konstansErteke() {
                echo self::allando;
        }
}
echo ProbaOsztaly::allando;       //ily módon is
                                    ➥elérhetjük
$osztaly = new ProbaOsztaly();
$osztaly->konstansErteke();       //ily módon is
                                    ➥elérhetjük
?>
```

Osztályok összehasonlítása

Az esetek jelentős részében szükségünk lehet arra, hogy megmondjuk két objektumról, hogy azok egyenlőek-e. A helyzet némiképp bonyolódott a *PHP 4*-hez képest, ugyanis ott csak a azonosság (===) operátort használtuk két objektum összehasonlítására. Két objektum mindig két külön memóriaterületet, azaz két külön „változót” jelentett. *PHP 5*-ben azonban bevezették, hogy az objektumokra referenciaként hivatkozunk. Ez máris szükségessé tette az összehasonlítás átalakítását: egyenlőséget és azonosságot egyaránt vizsgálhatunk

Az objektumok egyenlőségét az „összehasonlító” (==) operátorral ellenőrizhetjük. Ekkor a két objektum megegyezik, ha ugyanazon tulajdonságai ugyanazokat az értékeket tartalmazták, és ugyanannak az osztálynak voltak példányai (Ez a *PHP 4*-es megegyezőség feltétele is). Mivel azonban itt referenciákkal dolgozunk, szükségünk lehet szigorúbb ellenőrzésre is, ha azt akarjuk tudni, hogy két hivatkozás ugyanarra az objektumra mutat-e. Ehhez használjuk a már említett azonosság operátort. Két objektum akkor azonos, ha az adott osztály ugyanazon példányaira hivatkoznak mindketten. Alapesetben, ha egy objektumot átmásolok egy másikba, a két objektum „azonos” lesz. Ez jelentősen bonyolítja az esetleges „valódi” másolást, de a későbbiek folyamán megtanulhatjuk, hogy hogyan legyünk úrrá a problémán. A fentiek alkalmazásával egy sor fejlett osztályt készíthetünk az egyszerűbb problémák intelligens elhárítására. Fontos, hogy feleslegesen ne alkalmazzunk objektumokat, mert azzal csak elbonyolítjuk az egyszerű feladatot. Gondosan válasszuk meg, hogy mikor és milyen osztályokat készítünk. Nos, a cikknek ugyan végére értünk, de a mondanivalónknak messze nem. Hátra van még például az öröklődés – mint nagy témakör, az elvont osztályok, felületek létrehozása, a különleges tagfüggvények használata. Ezekkel és más egyéb érdekességékkel foglalkozunk cikksorozatunk következő epizódjában.

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

Hálózatok (12. rész)
Hidak

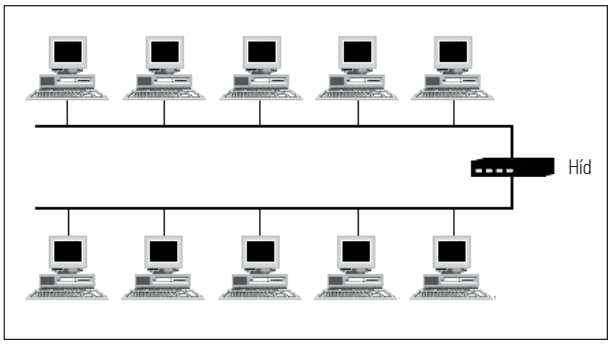
Mit tegyünk, ha több LAN-t szeretnénk egymással összekapcsolni? Ilyenkor hidakra van szükségünk, amelyekkel akár különböző szabványú hálózatokat is együttműködésre bírhatunk.

Minél több állomás van egy ***LAN***-on, az állomások annál nehezebben tudják megszerezni az osztott csatornát. Előbb-utóbb már annyira túlterhelt lesz az átviteli közeg, hogy a hálózat használhatatlanná válik az egyre gyakrabban kialakuló versenyhelyzetek, illetve ütközések miatt. A sávszélesség növelése persze jelenthet megoldást, de ez gyakran az összes hálózati eszköz lecserélésével jár, amely sosem olcsó mulatság.

Van azonban más lehetőség is. Az előző részben bemutattuk a ***kapcsolót (switch)***, amely külsőségeiben némileg hasonlít a hubokhoz, de a belső ennél sokkal többet rejt. A végpontokra ***(portok)*** itt is állomásokat kapcsolhatunk, de a tőlük érkező kereteket a kapcsoló – a hubokkal ellentétben – csak a címzett(ek)nek továbbítja. A kapcsoló tehát egy „intelligens" eszköz, képes belelátni a rajta átmenő adatokba. Mivel a keretek nem jutnak el oda, ahol senki sem kíváncsi rájuk (kivéve persze a hálózaton hallgatózókat, de ők most nem számítanak), csökken a csatorna terheltsége.

Egy kapcsoló végpontjaira ugyanakkor nem csak egyetlen munkaállomást köthetünk, hanem akár egy egész „***LAN***-nyit" is. Ettől kezdve a kapcsoló többé már nem kapcsoló, hanem ***híd (bridge)***. Itt aztán egészen új problémák merülnek fel, de még mielőtt ezekkel szembesülnénk, válaszoljunk meg egy kínzó kérdést: pontosan miért is van szükség hidakra? A kérdést egyből át is fogalmazzuk: miért lehet arra szükség, hogy egy szervezeten belül az állomásokat több különálló ***LAN***-ba foglaljuk, majd azokat hidakkal összekapcsoljuk, ahelyett, hogy összerakjunk egy jó nagy hálózatot, amelyhez minden állomás közvetlenül kapcsolódna?

Az első ok már sejthető: a terhelés megosztása. Ahogy azt az imént már kifejtettem ha sok az állomás, akkor a csatorna terhelt lesz. Ha nem szeretnénk terhelt csatornát, akkor meg kell növelnünk a sávszélességet. Nagy számú állomás esetében azonban óriási sávszélességre lenne szükségünk, amelyre nem biztos, hogy szert tudnánk tenni. Ilyenkor a hálózatot több részre ***(szegmensre)*** kell osztani, és valamiképp biztosítani azt, hogy a forgalom nagy része ne kerüljön ki a gerinchálózatra (a szegmenseket összekötő közegre), hanem maradjon a különálló részeken belül.

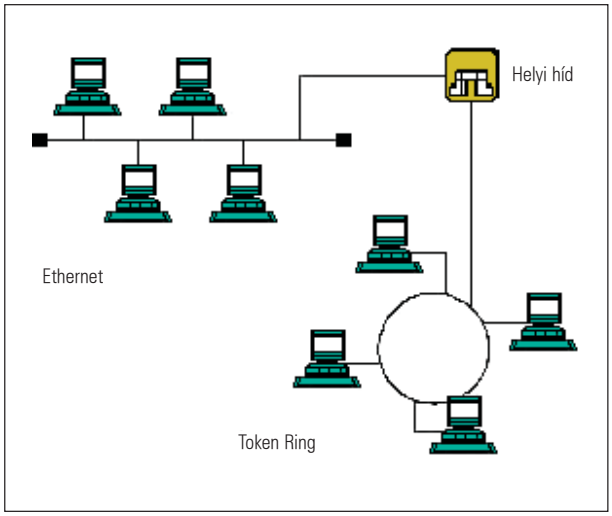


1. ábra

Miként lehet ezt megoldani? Tegyük fel, hogy van kismillió munkaállomásunk. A felhasználók az állományaikat a hálózat fájlszerverén tárolják. Mivel mindenki gyakran végez fájlműveleteket, ezért biztos, hogy a LAN a terhelés miatt használhatatlanná válik. Kivéve akkor, ha a munkaállomásokat szegmensekbe szervezzük, és minden szegmenshez tartozik egy saját fájlkiszolgáló. Mivel a szegmensekben viszonylag kevés állomás van, a csatorna ezeken belül már nem lesz különösebben terhelt.

Hogy miként is képzelhetjük el a szegmenseket, azt az *1. ábra* illusztrálja. A szegmensek különálló ***LAN***-ok, amelyek valamennyien saját csatornával rendelkeznek. Magukat a szegmenseket hidakkal köthetjük össze. Ha kevés szegmensünk van, akkor elegendő egyetlen kapcsoló, egy kiterjedt hálózat esetében azonban elképzelhető, hogy több kapcsolót kell egymással összekötnünk.

Hidakra akkor is szükségünk lehet, ha a hálózat amúgy bőven elbírná a terhelést. Előfordulhat például, hogy túl messze vannak egymástól az állomások. Bizonyos típusú ***LAN***-ok esetében fizikailag korlátozva van a hálózat megengedett mérete. Az ***Ethernet*** esetében ez például 2500 méter. Vagy nézzük azt az esetet, amikor két különálló épületben elhelyezkedő állomásokat szeretnénk egy hálózatba kötni. Sokkal megbízhatóbb, na és persze olcsóbb



2. ábra

megoldás az, amikor a két épületben két különálló ***LAN***-t készítünk, és azokat egymással hidak és infravörös átvitel segítségével kapcsoljuk össze, mint ha átvezetünk egy koaxiális kábelt.

A hidak másik fontos feladata a különböző szabványú hálózatok közötti átjárhatóság megteremtése *(2. ábra)*. Ilyen igény általában akkor szokott felmerülni, amikor már van két működő, ám teljesen eltérő típusú hálózat, és utólag jutott csak az üzemeltetők eszébe, hogy milyen jó is lenne, ha a két hálózat egymással is tudna kommunikálni. Van azonban olyan eset is, amikor direkt különböző szabványú hálózatokat szeretnének telepíteni egy szervezeten belül, mert bizonyos feladatokra az egyik alkalmasabb, mint a másik. Bármi is legyen a kiinduló ok, ezt a feladatot a hidaknak meg kell oldaniuk. Sejthető, hogy ez nem könnyű, hiszen például a vezérjeles gyűrű ***(token ring)*** más keretmérettel dolgozik, mint az ***Ethernet***, de sok más egyéb alapvető különbség is van.

Még egy dolog van, ami miatt áldott a hidak tevékenysége, ez pedig a megbízhatóság és a biztonság szempontjából érdekes. Az utóbbiról már volt szó az előző részben. Mivel az adatszórós hálózaton mindenki hallhat mindenkit, ezért könnyen lehallgathatóak olyan dolgok, amelyeknek sosem kellett volna más fülébe jutniuk. A hidak megnehezítik ezt a tevékenységet, ám teljes biztonságot nem nyújtanak. A hidak azonban okosak, így meg tudjuk mondani nekik, hogy milyen helyzetben mit továbbítsanak, illetve mit ne továbbítsanak. Ezzel megnövelik a hálózat stabilitását is, hiszen védelmet nyújtanak a meghibásodott állomás által folyamatosan a csatornára bocsátott szeméttől, illetve a hálózat megbénítását célul maguk elé tűző rosszakaróktól.

Hidak IEEE 802 szabványú hálózatok között

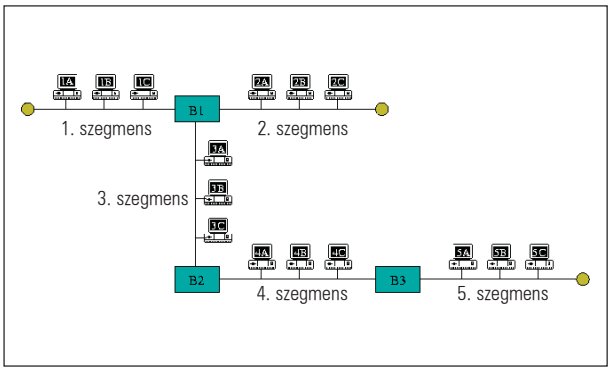
Még az ***Ethernet*** bemutatása előtt megemlítettük, hogy többféle ***LAN*** megvalósítás létezik. Ezek a megvalósítások mind szabványosítottak, azaz működésük a legapróbb részletekig dokumentálva van, és minden ilyen hálózat köteles betartani a szabványban leírtakat. A világ legnagyobb műszaki szakmai szervezete, az ***IEEE (Institute of Electrical and Engineers)*** többféle szabványt készített helyi hálózatokhoz, amelyek ***IEEE 802***-es szabványokként híresültek el. Az előző részben a ***802.3***-on ***(Ethernet)*** kívül bemutattunk más szabványokat is. Ezek a szabványok csupán a fizikai és az adatkapcsolati réteg megvalósításában különböznek, onnan felfelé kompatibilisek egymással.

A gond azonban pont az, hogy a különböző szabványok adatkapcsolati rétegei nem kompatibilisek egymással. A hidaknak tehát valahogy át kell tudniuk alakítani a rajtuk átmenő forgalmat úgy, hogy az a célhálózatban tovább tudjon haladni. Ha úgy tetszik, a híd egy olyan eszköz, amely biztosítja a keretek számára egy másik típusú hálózatba történő átjárást. Ez azonban egy nagyon veszélyes út.

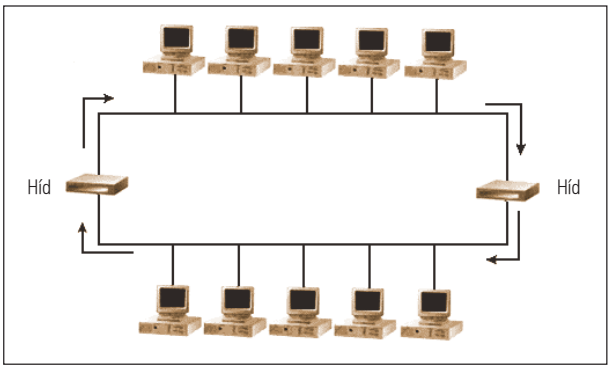
Rögtön az első probléma, hogy két összekapcsolt hálózat nem biztos, hogy ugyanolyan sebességgel üzemel. Amikor egy gyorsabb hálózat küldi keretek sorozatát a lassabb felé, akkor a híd nem tudja azt ugyanolyan gyorsan továbbítani. Amíg egy keret vár a továbbításra, addig a híd belső memóriájában kap helyet. Minden memória véges, ha a gyorsabb hálózatról folyamatosan érkeznek az adatok, akkor előbb-utóbb betelik a puffer, és a hídnak meg kell szabadulnia pár kerettől. Ugyanez lehet a helyzet, ha egy leterhelt ***Ethernet*** hálózat felé továbbítunk. A vezérjeles hálózatoknál a terhelés nem jelenthet puffer-túlcsordulást, hiszen biztosítva van, hogy a híd szabályos időközönként mindig adni tudjon.

Másik probléma, hogy a híd a hálózatban torlódási pont lehet. Ez nem csoda, hiszen minden szabvány más keretformátummal dolgozik, tehát a beérkező kereteket át kell alakítani és újból ki kell számolni az ellenőrző-összeget, ez pedig időbe telik. Miért baj ez? Tegyük fel, hogy a hálózati réteg egy hatalmas üzenetet ad át az adatkapcsolati rétegnek. Mivel a kereteknek van egy maximális méretük, ezért ezt az üzenetet több keretre kell szabdalni, majd azokat sorban továbbítani. Ezeket a kereteket a hídnak egyenként kell átalakítania. Kellően sok keret esetén az átalakításra fordított idő olyan nagy lehet, hogy a forrásállomás nyugtára váró időzítője lejár, és ismét elkezdi sorban küldeni a kereteket. Ezeket a hídnak megint csak át kell alakítania. Előfordulhat, hogy a forrás beleun az állandó ismételgetésbe, és hibaüzenetet dob, miszerint a célállomás nem válaszol. A probléma pedig nem is a címzettel van.

A legsúlyosabb bajok forrása azonban a különböző keretméretek. Nyílván addig nincs gond, amíg egy olyan hálózatba továbbítunk kereteket, ahol a maximális méret nagyobb, mint a küldő hálózatban. Fordított esetben azonban az ésszerű teendő az lenne, ha a túl nagy keretet feldarabolnánk. A probléma azonban az, hogy az adatkapcsolati protokollok nem támogatják a keretek összevissza szabdalását. Amikor egy állomás elküld egy keretet, akkor két esettel számol: vagy épségben megérkezik a célállomáshoz, vagy valahol elveszik útközben. De arra álmában nem gondol, hogy egy híd feldarabolja. Hiszen akkor ő egyet küldött, de a cél kettőt kap. Persze nem lenne túl nehéz ezeket a protokollokat alkalmassá tenni a feldarabolt keretek kezelésére, csak épp a ***802***-es szabványban ez nem szerepel. Így nem létezik megoldás. Illetve létezik: ha valaki egy olyan keretet küld, amelyet



3. ábra



4. ábra

a célhálózat nem tud elfogadni, akkor a híd egyszerűen eldobja. Ez azonban nem túl megnyugtató megoldása a problémának.

Ha ***802.4 (vezérjeles sín)*** hálózatról adunk ***802.3*** felé, akkor két további probléma is felmerül. Az első természetesen a prioritás. Az ***Ethernet*** hálózatokban az állomások nem rendelkeznek prioritással, míg a ***802.4*** esetében van ilyen szolgáltatás. Persze ez csak akkor jelenthet problémát, ha a két, egymással kommunikáló ***802.4***-es hálózat között van egy ***802.3***-as is. Ebben az esetben sajnos a keretek prioritását nem lehet megőrizni.

Másik megoldhatatlan problémáért az ideiglenes vezérjelátadás nevű szolgáltatás okolható. Ez arra való, hogy ha egy ***802.4***-es hálózatban küldünk egy keretet, akkor a célállomásnak lehetősége nyílik arra, hogy visszakapja a vezérjelet ahhoz, hogy a nyugtát visszaküldhesse. Amikor egy ilyen keret érkezik a hídhoz, akkor annak komoly lelkiismereti kérdéssel kell megbirkóznia: vagy hazudik, és visszaküld egy hamis nyugtát a feladónak, vagy becsületes módon nem szól semmit, csak továbbítja a keretet. Mind a kettő nagyon veszélyes, nehéz eldönteni, hogy melyik a jobb megoldás. Átverni a feladót nem bölcs dolog, főleg úgy, hogy elképzelhető, a célállomás nem is működik. A keretet nem nyugtázni is merész húzás, hiszen a feladó megelégelheti a dolgot egy „célállomás halott" hibaüzenettel. Tökéletes megoldás erre sem létezik.

Amikor ***802.3***-ról ***802.4***-re szeretnénk egy keretet átjátszani, akkor ott csak a már említett prioritás okozhat gondot. Pontosabban az, hogy milyen prioritást állítsunk be az átküldött kerethez. Talán a legjobb taktika az, ha mindig a legmagasabb fokú prioritást rendeljük az átmenő keretekhez, mivel feltehetően az már úgyis késleltetésben van.

Ha két ***802.4***-es hálózatot szeretnénk egy híddal összekötni, akkor szintén szembetaláljuk magunkat az ideiglenes vezérjelátadás problémájával. Ilyenkor a hídnak nagy szerencsejátékosnak kell lennie. Ha ugyanis úgy hozza a sors, hogy a keretet egyből tudja továbbítani, akkor van rá esély, hogy a nyugta időben megérkezhessen. Ha mégsem alakulnak úgy a dolgok, akkor csalni kell: a keret prioritását meg kell növelni. Ezzel lerövidíthetjük a nyugta célbaérkezésének idejét.

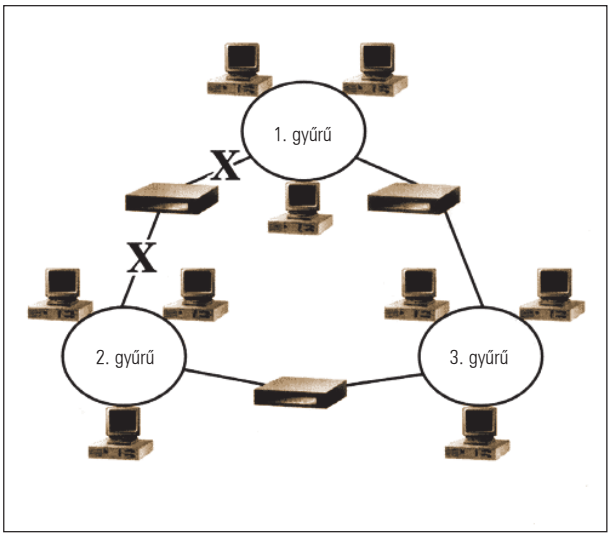
Nem kérdés, hogy elég sok problémával kell szembenéznünk akkor, amikor két hálózatot egy híddal szeretnénk összekötni. De mi a helyzet akkor, ha nem csak két, hanem több ***LAN***-t és több hidat tartalmazó összekapcsolt hálózatunk van? Nos, erre is létezik ***IEEE*** szabvány, de hasonlóan a hálózatokhoz, itt is több (egész pontosan kettő) van, amelyek természetesen nem kompatíbiliesek egymással.

Transzparens/feszítőfás hidak

Ezek a hidak abban a szellemben készültek, hogy bármiféle hardveres, illetve szoftveres beállítás nélkül telepíthetőek legyenek. Ezt szó szerint úgy képzelhetjük el, hogy amint bekötjük a hidat a hálózatba és bekapcsoljuk, a rendszerünk máris működik. Nem kell sem a hidat, sem az állomásokat felkonfigurálnunk. Azért nevezzük ezeket transzparens ***(átlátszó)*** hidaknak, mert az állomások számára láthatatlanok, ők ugyanúgy látják egy másik hálózat állomásait, mintha közös ***LAN***-ban lennének. Amikor egy keret megérkezik, nem tudhatjuk, hogy útja során hány hídon ment keresztül, azaz hány híd van köztünk, és a velünk kapcsolatban lévő állomás között.

Hogy megérthessük a transzparens hidak működését, tekintsünk a *3. ábrára*. Itt négy ***LAN***-t látunk három híddal összekötve. Tegyük fel, hogy az ***1A*** gép keretet küld az 1B-nek. Mivel ezek egy szegmensen vannak, így a ***B1*** híd a keretet biztosan eldobja. Most nézzük azt az esetet, ha a keret címzettje ***3B***, amely egy másik szegmensen van. A hídnak most továbbítania kell a keretet, csak az a kérdés, hogy hova. Ilyenkor a memóriájában lévő táblázatból kiolvassa, hogy a keret címe alapján melyik végpontjára kell átjátszani. Látja, hogy a ***3B*** gép a hármas szegmensen van, így továbbítja a megfelelő portra. Fontos, hogy a táblázat nem azt mondja meg, hogy melyik szegmensen van a célállomás, hanem azt, hogy merre kell irányítani. Ha a címzett az ***5C*** lenne, akkor is ezen a porton menne tovább a keret, csak át kell még mennie két másik hídon. De ezzel a ***B1*** hídnak nem kell foglalkoznia.

De azt mondtuk, hogy a transzparens hidak semmiféle beállítást nem igényelnek. Mégis honnan tudják, hogy merre kell továbbítani az egyes kereteket? Valóban, kezdetben az összes híd táblázata üres. Ha tehát jön egy olyan keret, amelyiknek a címzettjét nem ismerik, nem tehetnek mást, mint hogy az összes kimeneten továbbítják (kivéve azt a portot, amelyikről a keret érkezett). Ezt úgy is mondjuk, hogy a hidak elárasztják a hálózatot. Ezután megjegyzik,



5. ábra

hogy ennek a keretnek ki volt a feladója, és melyik ***LAN*** felől érkezett. Ha a későbbiekben egy olyan keret érkezik, amelynek a címzettje a kérdéses állomás, akkor már tudni fogják, hogy azt melyik kimenet felé kell irányítani. Ahogy egyre több keret érkezik, úgy fogják ismerni egyre részletesebben a hálózat felépítését. Ezt az algoritmust hátrafelé tanulásnak nevezzük.

Egy hálózat felépítése azonban ritkán állandó, a lelkes rendszergazdák egyfolytában átalakítják. Például új hidakat telepítenek, régieket távolítanak el, és az állomásokat az egyik ***LAN***-ból a másikba költöztetik. A hidaknak ezért folyton naprakész (sőt percre pontos) információval kell rendelkezniük a hálózat aktuális állapotáról. Ezért fontos tudniuk, hogy egy adott cím melyik időpontban került a táblába. Ha érkezik egy keret, amelynek forrása már szerepel a táblában, akkor annak időpontját a híd az aktuális időpontra írja felül. Ezenkívül minden, néhány percnél öregebb bejegyzést a hidak törölnek. Ezzel elérhető, hogy ha egy állomást elköltöztetünk a hálózat egy teljesen más pontjába, akkor is perceken belül visszaáll a normális működés.

Tekintetünket szegezzük most a *4. ábrára*, ahol csupán csak két ***LAN*** van, viszont a rendszergazdák biztosra akartak menni, és két híddal kötötték össze a szegmenseket, hogy meghibásodás esetén se legyen semmi probléma. Tegyük fel, hogy a felső ***LAN*** egyik állomása egy olyan keretet bocsát útnak, amelynek a címzettje mindkét híd számára ismeretlen. Nincs mit tenni, mindketten elárasztanak, azaz a kérdéses keretet mindketten az alsó ***LAN***-ra másolják. Ezután nem sokkal a bal oldalt lévő híd felfedezi a jobb oldali híd által átjátszott keretet, amit ő visszamásol a felső ***LAN***-ra, amit majd ismét a jobb oldali fog ismét az alsó ***LAN***-ra másolni. Ennek a folyamatnak sohasem lesz vége.

Mitől alakulhatott ki ez a roppant kellemetlen helyzet? Hát azért, mert hurok volt a hálózatunkban. Ezért a hidaknak valahogy egyeztetniük kell egymással, és logikailag úgy átalakítani a hálózatunk topológiáját, hogy egy LAN-hoz csak „egy út vezessen”.

Képzeljük el úgy a hálózatunkat, mint egy olyan gráf, amelynek a csúcsai a ***LAN***-ok. Erre láthatunk példát az ***5. ábrán***, ahol a változatosság kedvéért vezérlőjeles gyűrűs hálózatokat kötöttünk össze egymással. Két csúcs akkor legyen egymással összekötve, ha össze vannak kapcsolva híddal. Jelen esetben minden csúcs össze van kötve mindenkivel. A feladat az, hogy egy körmentes, ám továbbra is összefüggő gráfot kapjunk, amely tartalmazza az eredeti gráf összes csúcsát. Ez az eredeti gráf úgynevezett ***feszítőfája (spanning tree)***. A feszítőfára az jellemző, hogy minden csúcsból minden csúcsba pontosan egy út vezet. Egy ilyen hálózati topológiában nem fordulhatna elő az előbb említett eset.

A hidaknak tehát egymás között meg kell állapodniuk a feszítőfában. Ehhez maguk közül ki kell választani valakit, aki a fa gyökerét fogja képezni. Erre egy elég frappáns megoldást alkalmaznak: minden híd megmondja a saját egyedi sorozatszámát. Akié a legkisebb, az lesz a gyökér. Ezután meg kell határozni azt a fát (a feszítőfát), amely minden ***LAN***-hoz a lehető legrövidebb utat tartalmazza. Ezt a fát természetesen újra kell számolni, ha a hálózati topológia megváltozik, például új híd kerül telepítésre.

A feszítőfát kiszámoló osztott algoritmus egyébként sohasem áll le, így a feszítőfa azonnal érzékelhető.

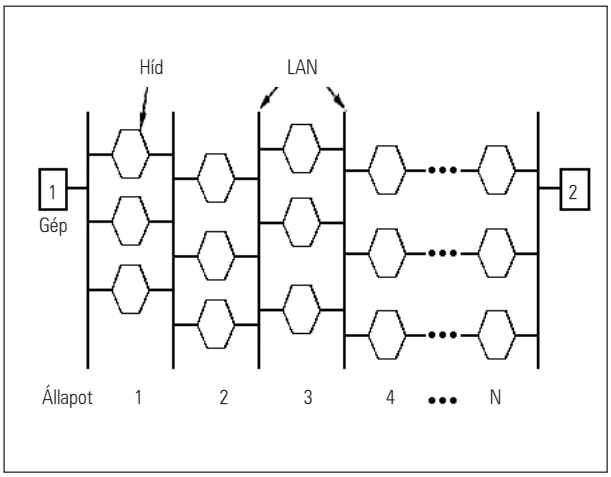
Az így kialakított új hálózati topológiánkban az összes ***LAN*** biztosan benne marad, de nem biztos, hogy az összes híd is! Amikor a fizikai topológiát ábrázoló gráfból egy élet elhagyunk, az azt jelenti, hogy az adott élhez tartozó hídnak blokkolnia kell azt a két kimenetét, amely az élhez tartozó két csúcsot (LAN-t) összeköti. Ha egy híd minden olyan használatban lévő portját blokkolta, akkor ő nem vesz tovább részt az adatforgalom továbbításában. Ez a helyzet az *5. ábrán* is. Az ott látható gráfból egyféleképpen készíthetünk feszítőfát: egy tetszőleges élt törlünk, azaz megszüntetjük a kapcsolatot két tetszőleges ***LAN*** között. Jelen esetben az 1. és a 2. között, de bármelyiket is választjuk, egy híd mindenképp munka nélkül marad.

Látható tehát, hogy a transzparens hidak képtelenek kihasználni a teljes rendelkezésre álló sávszélességet. Nem úgy mint a következő hídcsoport.

Forrás által irányított hidak

Ezek aztán tényleg kihasználják a sávszélességet, habár van egy buktatója a dolognak, de erről majd kicsit később. A működési elv azzal a nehezen teljesíthető feltételezéssel él, hogy minden állomás, aki üzenetet szeretne küldeni egy másik állomásnak, pontosan tudja, hogy a címzettel egy szegmensen van-e, vagy sem. Ha nincs, akkor viszont tudja, hogy miként lehet (mely hidakon keresztül) eljutni abba a hálózatba.

A ***forrás általi forgalomirányítás (source routing)*** könnyebb megértéséhez ismét segítségünkre lesz a 3. ábra. Ha az ***1A*** szeretne mondani valamit az ***1B***-nek, akkor a keret címének utolsó helyiértékű bitjét 0-ra állítja, mivel egy szegmensen vannak. Ekkor a ***B1*** híd nem foglalkozik a kerettel. Ha azonban az ***5C*** gépnek szeretne küldeni, akkor ennek a bitnek az értéke 1 lesz, és a keret fejlécébe beszúrja a pontos útvonalat is. Ahhoz, hogy ezt az útvonalat le lehessen írni, minden ***LAN***-nak rendelkeznie kell egy egye-



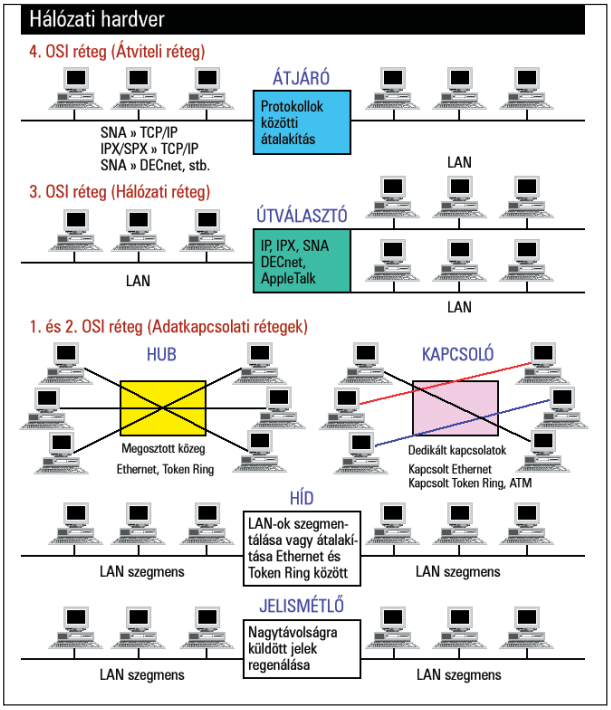
6. ábra

di címmel. Ugyanez a helyzet az összes ugyanahhoz a szegmenshez kapcsolódó híddal is. Tehát a ***B1*** és ***B2*** hidak címének különbözőnek kell lennie, de a ***B1*** és ***B3*** címe megegyezhet. Az útvonal tehát egy olyan sorozat lesz, amelyben felváltva szerepel egy híd és egy ***LAN*** címe. Az ***1A – 5C*** útvonal tehát a következőképp fog kinézni: ***B1***, ***Seg3***, ***B2***, ***Seg4***, ***B3***, ***Seg5***. Amikor egy híd megkap egy ilyen keretet, akkor belenéz az előírt útvonalba, és megkeresi benne annak a ***LAN***-nak a címét, ahonnan a kérdéses keret érkezett. Ezután megnézi, melyik híd címe áll mögötte. Ha ez a cím a saját címe, akkor továbbküldi az előírt szegmensre.

A kérdés már csak az, hogy honnan tudhatják az állomások azt, hogy a cél felé pontosan milyen út is vezet. Sőt, nem elég bármelyik útvonalat ismerni, ahhoz hogy a dolog valóban hatékony legyen, a legeslegrövidebb utat kell megtalálniuk.

Amikor egy állomás nem ismeri a címzettjének pontos helyét, akkor egy úgynevezett ***felkutató keretet (discovery frame)*** bocsát útnak. Ezeket adatszórással továbbítja, és minden híd minden irányba továbbítja, tehát biztosan megérkezik az összes ***LAN***-ra. Amikor a címzett rádöbben, hogy valaki holléte felől érdeklődik, egy válaszkeretet küld vissza. Ahogy ez a keret visszafelé indul, minden olyan híd, amelyen áthalad, beleírja azonosítószámát. Így amikor visszaér, az állomásnak lehetősége nyílik megvizsgálni a visszafelé vezető útvonalat, amelyből könnyedén kiszámítható a célhoz vezető legoptimálisabb útvonal.

A már említett buktató akkor jelentkezik látványosan, ha a hálózatunk topológiája hasonló a *6. ábrán* bemutatotthoz. Itt a ***LAN***-okat sorba egymás után kötöttük, minden egyes ***LAN***-t három híd kapcsol össze egymással. Az 1-es állomás a 2-es pozíciójáról szeretne többet megtudni, ezért kutató keretet indít útnak. Mivel ezeket a kereteket az összes híd továbbítja, ezért a szomszédos LAN-on már három kutató keret lesz jelen, a 3-on kilenc, a 4-en pedig 27. Ez bizony exponenciális ütemben zajló növekedés, tehát 10 ***LAN*** esetében majdnem 20 ezer, 20 ***LAN***-nál pedig több mint 1 milliárd keretet jelent. Talán nem kell ecsetelni, hogy ez mekkora terhelést is jelent. Ezért is hívják ezt a jelenséget keretrobbanásnak.



7. ábra

Felmerülhet a kérdés, hogy a transzparens hidakat miért nem veszélyezteti a keretrobbanás, miközben ők is ugyanezt csinálják azokkal a keretekkel, amelyeknek a címzettje ismeretlen. Valójában ott is ugyanez a folyamat játszódik le, csak nem árasztják el velük az egész hálózatot, hanem kizárólag a feszítőfa mentén küldik tovább. Ez pedig csak lineáris ütemben történő növekedés, amely nem jár ilyen drasztikus végeredménnyel.

Hálózati eszközök – összefoglalás

Sorozatunk e részének végén érdemes egy gyors összefoglalást végezni azzal kapcsolatban, hogy milyen hálózati eszközöket is ismerünk idáig, és ezeknek pontosan mi a feladatuk. Ehhez hasznos segítséget nyújt a *7. ábra*. Az ***átjáró*** (***gateway***) és az ***útválasztó*** (***router***) még ezidáig nem szerepelt, mivel ezek a felsőbb, eddig még nem tárgyalt rétegekben tevékenykednek. Fontos, hogy hiába végez a híd forgalomirányítást, illetve üzemel „átjáróként” különböző szabványú hálózatok között, nincs köze az útválasztóhoz és az átjáróhoz.

A hub a fizikai rétegben tevékenykedik és nagyon buta eszköz: ami az egyik portján bemegy, az kijön a többin.

A kapcsoló (switch) ennél sokkal okosabb, hiszen képes az adatkapcsolati réteg adategységeit, a kereteket olvasni, így csak arra a kimenetre küldi az adatot, ahol a címzett található. Nem így a jelismétlő (repeater), amely a hubbal van „egy szinten”: ő is csak ismételni tudja a bejövő jeleket, igaz, felerősíti azokat.

A következő részben a nagy sebességű hálózatokról és műholdakról lesz szó.

Garzó András
garzo@interware.hu

Fénysebességgel

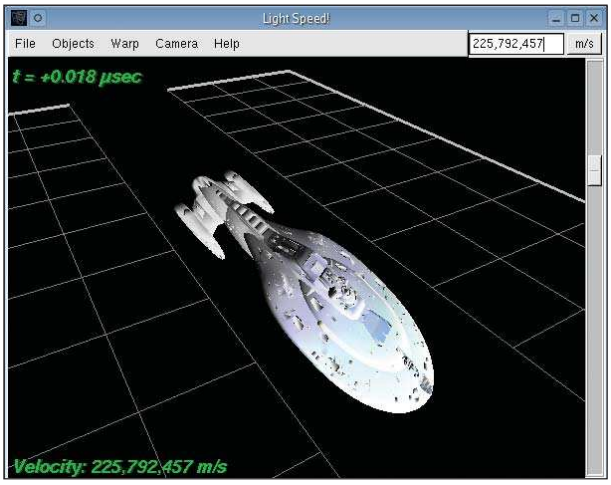
Most már olyan űrhajót is tudunk tervezni, ami a vöröseltolódás után is jól néz ki. Relativitás, csillagok és sok minden más.

Igazad van, ***François***, a számítógépek és az operációs rendszerek komoly utat jártak már be. Nemcsak az a szerencse ért bennünket, hogy már ma használhatjuk a jövő operációs rendszerét, de minden eddiginél gyorsabb gépek tudását élvezhetjük. Mit is mondjak, talán némi elfogult szeretettel gondolok vissza első ***x86*** alapú számítógépemre. Egy felturbózott ***XT*** volt, a processzora kereken 10 MHz-en ment. Kisebb vagyont költöttem arra, hogy a memóriáját 640 kB-ról 1024 kB-ra bővítsem; emlékszem, egy maroknyi lapkát kellett beledugdosnom az alaplapon lévő foglalatokba.

Quoi? Természetesen, ***mon ami***, hiába volt jó kis gép akkoriban, nem kívánok lemondani a kor vívmányairól. Az olyan lenne, mintha a Linuxot másik operációs rendszerre cserélném. Tudod, François, érdekes elmélázni azon, hogy milyen messzire jutottunk – a megahertzes processzorokból gigahertzesek lettek, alig néhány év alatt! Vajon mi lesz majd tíz év múlva? Igen, lehet, hogy túllépjük a fénysebességet is, de attól tartok, ***mon ami***, az azért egy kicsit még várat magára. Viszont adtál egy jó ötletet.

Mon Dieu! Már itt is vannak a vendégeink! A borospincébe, ***immédiatement***! Indíts az északi szárnyba, ott van a legújabb szállítmány ***Bordeaux***-ból, ***Henri*** tegnap hozta. Találsz ott néhány láda 2000-res ***Châteauneuf-du-Pape***-ot is. Bocsássatok meg, ***mes amis***. Foglaljatok helyet, helyezzétek magatok kényelembe. ***François*** és jómagam arról csevegtünk, hogy az utóbbi években mekkorát fejlődött a világ. Készséges felszolgálóm felvetette a fénynél is gyorsabb számítógépek ötletét, mely talán mai témánk, a nagyteljesítményű számítások mindenható eszköze lenne. Hiába lennének azonban az adatokat a fénysebességnél is gyorsabban szállító gépeink, mi magunk továbbra is képtelenek lennénk akkora információmennyiséget befogadni. Egy biztos, a legújabb ***Linuxvilág*** olvasgatása közben nem fogjuk látni összecsúszni a csillagokat.

Sebesség tekintetében a fényt semmi sem győzheti le, hacsak valamiféle rejtélyes anyag-antianyag motorral és némi dilítium kristállyal nem. Ugyanezt az érzést élhetjük át, ha beállítunk magunknak egy képernyőkímélőt, és az ***xscreensaver*** kínálatából a ***rocks***, azaz sziklák nevűt választjuk, esetleg az ***OpenGL*** alapú ***space***, űr nevűt a ***KDE*** saját listájáról. Az, hogy mit látnánk a fénysebesség megközelítése közben, a tudományos-fantasztikus filmek örök témája,



1. ábra Vajon mi történne egy űrhajóval, miközben megközelítené a fénysebességet?

ám nagyon valószínű, hogy a valóságban soha nem tudjuk meg. Éppen ez adta az ötletet ***Daniel Richard G. Light Speed!*** nevű programjához, amely azt szemlélteti, hogy mit látnánk, ha egy tárgy megközelítené a fénysebességet.

A program számos valódi hatást vesz figyelembe, mint például a ***Lorentz***-összehúzódás, a vörös/kék ***Doppler***-eltolódás, az orrlámpa-effektus és az optikai elhajlás. A ***Light Speed!*** webhelyének ***About*** lapján mindegyik hatás leírását megtaláljuk.

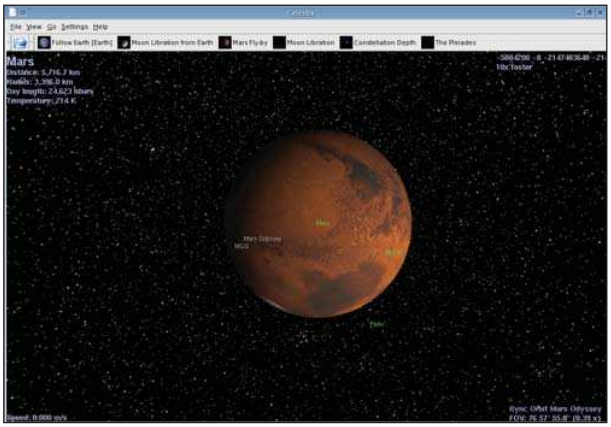
Mes amis, biztos vagyok benne, ez a bor ízleni fog – valóban testes ital, fekete gyümölcsök aromájával, sejtésnyi kávéval és hosszan tartó utóízzel. Kortyolgassátok, amíg mi felgyorsítunk fénysebességre. A fordítás rendkívül könnyű. Jó, ha tudjuk, szükségünk lesz az ***OpenGL*** vagy ***Mesa*** fejlesztői könyvtárakra, valamint a ***gtkglarea*** könyvtárakra. Innen már csak egy egyszerű kitömörítés és lefordítás következik öt lépésben:

```
tar -xzvf lightspeed-1.2a.tar.gz
cd lightspeed-1.2a
./configure
make
su -c “make install”
```

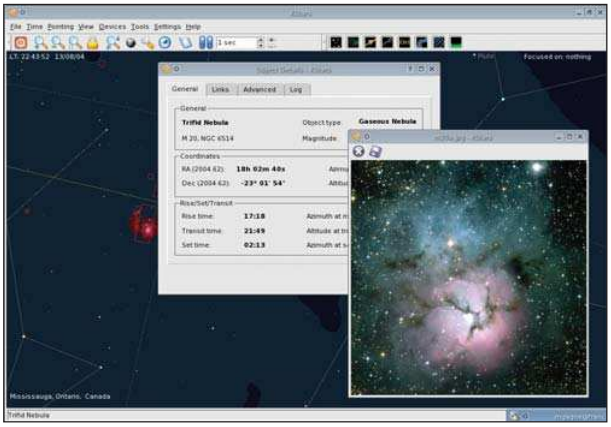
A program indításához a `lightspeed` parancsot kell kiadni. Egy ablaknak kell megjelennie, benne pedig egy három dimenziós rácskockának. A jobb felső ablakban lévő mezőbe lehet beírni a sebességet, méter/másodpercben. Mérsékelten magas értékkel indítsunk, később a felfelé és a lefelé nyíllal növelhetjük és csökkenthetjük a sebességet. Ha lenyomjuk az ***Entert***, a tárgy a megadott sebességre gyorsul, az ebből fakadó hatásokat pedig az ablakban követhetjük figyelemmel. Egy a fénysebesség közelébe kerülő, fokozatosan eltorzuló kocka roppant érdekes dolog, ám ha a menüsorban a ***File (Fájl)*** majd a ***New Lattice (Új rácsozat)*** parancsra kattintunk, végül megadjuk a három dimenzióbeli pontok számát, akkor bonyolultabb alakzatokat is előállíthatunk. A valódi kérdés az, vajon mi történne egy űrhajóval, ha megközelítené a fénysebességet? Micsoda szerencse, hogy itt van nekünk a ***Light Speed***! A webhelyről objektumokat is le lehet tölteni, érdemes megpróbálkozni vele. Három objektumot találunk, köztük a ***Star Trek Voyager*** (*1. ábra*) modelljét. Ha másik modellt szeretnénk kipróbálni válasszuk a ***File*** (***Fájl***) menü ***Load Object*** (***Objektum betöltése***) pontját.
Még jobban szórakozhatunk, ha ellátogatunk a ***3-D Cafe*** oldalra (lásd a forrásokat), ahol rengeteg további három dimenziós modellt és hálót találunk. Ne csak űrhajókat próbáljunk ki, egy fénysebességgel haladó versenyautó legalább olyan érdekes. Ne feledjük, hogy csak ***3DS (3D Studio)*** vagy ***LWO (LightWave 3D)*** kiterjesztésű modelleket tudunk használni. Remek dolog eljátszani a gondolattal, hogy mi történne ilyen körülmények között, de legalább ennyire szeretne mindannyiunk tíz warp sebességgel száguldani az űrben, bámulni az egymásba tolódó csillagokat, majd még a következő borosüveg kiürülése előtt megérkezni egy távoli világba. Ilyesfajta kirándulást a ***Chris Laurel***-féle ***Celestiával*** tehetünk. A ***Celestia*** segítségével többek közt felfedezhetjük saját naprendszerünket, több mint százezer csillagra látogathatunk el és megismerhetjük a Földről indított különféle űrszondák sorsát. Az oldalról forrást is letölthetünk, de ***Mandrake***, ***SuSE*** és egyéb terjesztések alá binárisokat is találunk. Ha nem találjuk a saját rendszerünkhöz való binárisokat, akkor sincs gond. A program ***OpenGL*** alapú, ezért szükségünk lesz a 3D könyvtárakra, de a fordítás ismét csak az öt lépésből álló, gyermekien egyszerű folyamat:

```
tar -xzvf celestia-1.3.1.tar.gz
cd celestia-1.3.1
./configure
make
su -c “make install”
```

Futtatásához a parancssorból vagy a parancsindítóból adjuk ki a `celestia` parancsot. Néhány billentyű szerepét nem árt ismerni, így ugyanis sokkal érdekesebb lesz az utazás. Az L betűvel tízszeresen gyorsíthatjuk az időt. Ekkor űrbéli utazásunk a hivatkozási pontként kiválasztott tetszőleges objektumhoz lesz viszonylagos. A K lenyomásával lelassíthatjuk az időt, ha túl gyorsan haladnánk. Az ALT-C billentyűkombináció a ***Celestia*** böngészőjét indítja el, amellyel különféle objektumok közül tudunk választani. A képernyő alján négy választógomb jelenik meg. Kattintsunk a ***With planets (Bolygókkal)*** gombra, ekkor megjelenik az ismert bolygókkal rendelkező csillagok listája. Szeretnénk ellátogatni az ***51 Pegasi*** körül keringő bolygóra? Kattintsunk az egér jobb gombjával az objektum nevére, válasszuk a ***Goto (Ugrás)*** parancsot, majd csatoljuk be az öveket! A fénynél is gyorsabban juthatunk el ebbe az idegen világba. Ha már ott vagyunk, kattintsunk rá az ***51 Pegasi*** csillagra, válasszuk a ***Follow (Követés)*** parancsot – így a csillagnál maradva vizsgálhatjuk meg a bolygó pályáját.



2. ábra A Mars körül keringő űrszonda követése



3. ábra A KStars online adatbázisokból – mint amilyen Hubble-é – is képes képeket beemelni

A gombokkal a csillagok megjelenítési módját is megadhatjuk, az apró tűszúrásoktól kezdve a kisméretű foltokon keresztül egészen részletesen megrajzolt korongokig változtathatjuk a képüket. Ha az összes billentyű szerepét meg szeretnénk ismerni, akkor kattintsunk a ***Settings (Beállítások)*** menü ***Configure Shortcuts (Gyorsbillentyűk beállítása)*** parancsára.
A ***Celestia*** nagyszerű eszköz, ha kényelmesen ücsörögve szeretnénk felfedezőútra indulni; valóban megéri letölteni. A csillagokon és a bolygókon túl a közeli égitestek körül keringő űrszondákra, például a ***Mars Global Surveyorra*** is ellátogathatunk. Próbáljunk az űrszonda felé fordulni, kattintsunk a Marsra, majd válasszuk a ***Follow (Követés)*** parancsot. (*2. ábra*) Most gyorsítsuk fel az idő múlását. Az adatbázisban számos nagyobb aszteroida is szerepel, ha gondoljuk, akkor ellátogathatunk például az ***Erosra***.
Ha a fénysebességen túli űrbéli utazgatás túlságosan felforgatja a gyomrunkat, esetleg hatására arcunk egészen újszerű, zöldes árnyalatot vesz fel, talán próbáljunk inkább a Föl-

dön maradva foglalkozni a végtelen rejtelmeivel. Miért nem biztosan álló székünkből szemléljük a csillagokat és bolygókat? A legkönnyebben ezt ***Jason Harris KStars*** programjával tehetjük meg. A ***KStars*** egy igazi planetáriumot varázsol asztalunkra. Mivel a ***KStars*** része a ***KDE*** környezetnek – a ***kdeedu*** csomagban található –, nem kell túl sokat keresgélnünk, hogy rábukkanjunk. A csomaggal kapcsolatos újdonságokról a weblapról tájékozódhatunk.

A ***KStars*** egyszerűen lenyűgöző, de ne gondoljuk, hogy egyszerű játékszer. A bolygók, 130 ezer csillag, 13 ezer mélyűrbéli objektum (bolygók és aszteroidák) adatbázisával a ***KStars*** valóságos csillagászati kincsesláda. Segítségével könnyedén beazonosíthatjuk az éjszakai égbolton látható csillagok, galaxisok, csillagködök és egyebek helyzetét. Pontosan beállíthatjuk, hogy mit akarunk megjeleníteni, ráközelíthetünk az objektumokra és – nekem ez a kedvencem – hálózati forrásokból képeket is tölthetünk le, például a ***Hubble*** vagy a ***Space Telescope Science Institute*** adatbázisából. Elég az egér jobb gombjával a kiszemelt objektumra kattintanunk, a felbukkanó ablakban megtekinthetjük részletes adatait, valamint nagyfelbontású képekre mutató hivatkozásokat is kapunk, ha vannak ilyenek. A *3. ábrán* a ***KStars*** képernyője látható, miközben a ***Trifid Nebula*** adatait jeleníti meg.

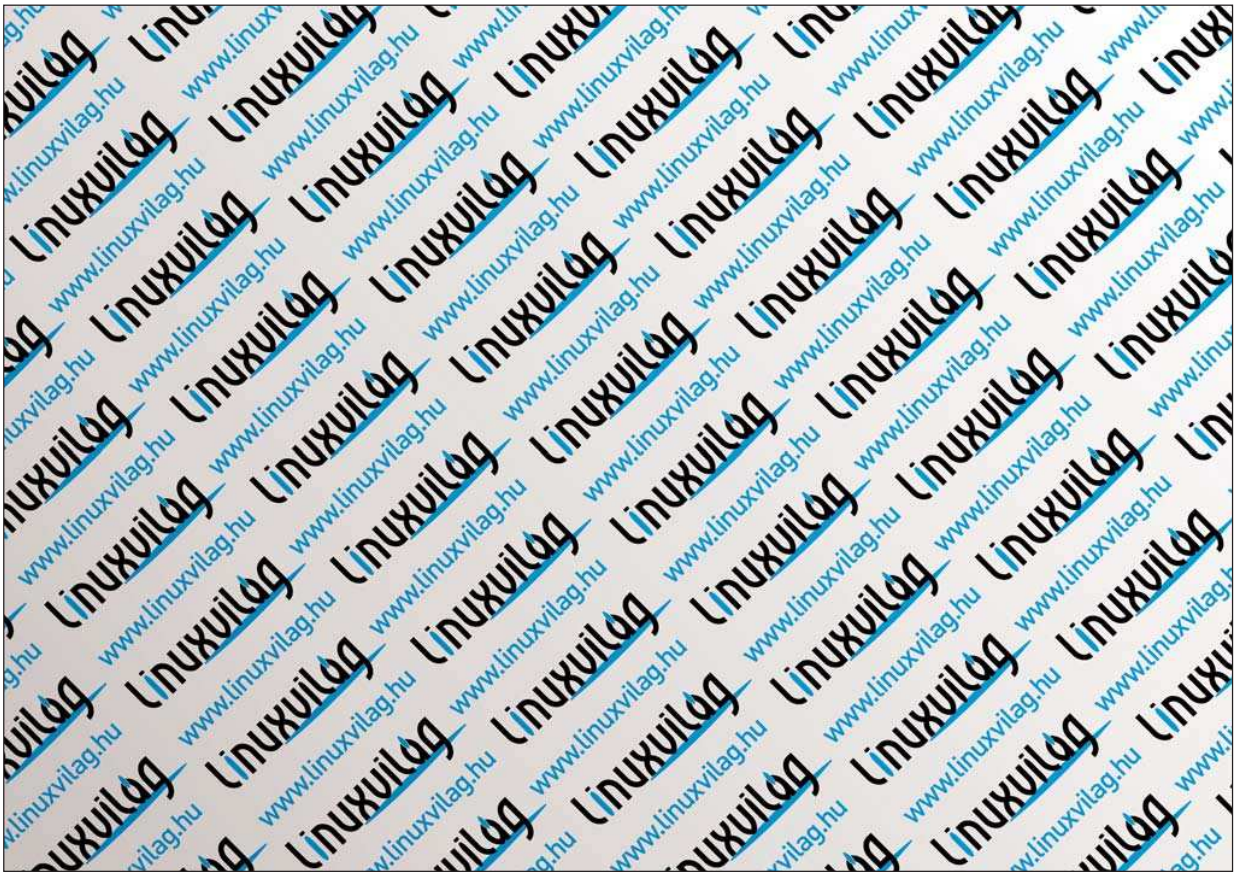
A ***KStars*** indításkor feltételezi, hogy az angliai Greenwichben vagyunk. Nyilván ezen változtatni szeretnénk, ehhez kattintsunk a ***Location (Hely)*** menü ***Geographic (Földrajzi)*** parancsára. Egy párbeszédpanel jelenik meg, egy világtérképpel kiegészítve. Kattintsunk nagyjából arra a területre, ahol élünk. Ekkor a térképtől jobbra földrajzi pontok egy listája jelenik meg. Válasszuk ki a megfelelőt, majd kattintsunk az ***OK*** gombra. Ha tudjuk, hogy pontosan mi lakhelyünk hosszúsági és szélességi koordinátája, akkor az ablak alján ezt is megadhatjuk.

A ***KStars*** jóval többet tud, mint amiről egy ilyen rövid látogatás alatt szólhatnék. Képes például saját teleszkópunkat vezérelni, a kívánt objektumokra irányítani és azokat követni. Akit érdekel a csillagászati fényképezés, annak megemlíteném, hogy a ***KStars*** a ***CCD***-k kezelésére is alkalmas. Noha ez a szolgáltatás jelenleg csak a ***Finger Lakes Instruments*** eszközeire terjed ki, a támogatást hamarosan bővíteni fogják.

Mon Dieu! Bár a fény sebességét meg sem közelítettük, mégis gyorsan megtörtént. Igen, az órára gondolok, ***mes amis***, ami bizony már zárórát mutat. Felfoghatatlan sebességekről beszélünk, de közben ne feledjük el, milyen csodás egy holdfényes éjszakán hátradőlni, és lassan kortyolgatni ezt a kiváló ***Châteauneuf-du-Pape***-ot. A következő alkalomig igyunk egymás egészségére, ***mes amis***! A vôtre santé! Bon appétit!

Linux Journal 2004. november, 127. 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.



A vagyoni jogi jogosultságok, avagy „pénz beszél...”

... a jogász meg mondja a magáét... mely szerint a vagyoni jogi jogosultságokkal tisztában lenni, minden szerző elengedhetetlen kötelezettsége.

A szerző joga, hogy a mű felhasználására bárkinek engedélyt adjon, ezt szoftver esetében általában egy – a számítástechnikai berkekben licenc néven ismert – felhasználási szerződés keretében teheti meg[1].

A vagyoni jogi jogosult

Tételezzük fel, hogy legújabb – kereskedelmi forgalomban szerzett[2] – ***„Handabanda”*** névre hallgató, füllentés-analizáló szoftverünk licencében azt találjuk, hogy a velünk szerződő jogosult nem ***Svindli Jimmy*** (aki egyébként a szerző), hanem a ***Víg Matróz és Társa Részvénytársaság*** (a továbbiakban ***VMT Rt.***). Ennek az alábbi okai lehetnek:

1. ***Jimmy*** a ***VMT Rt.*** munkavállalója, így az általa – munkaviszonyban – alkotott szoftverek vagyoni jogai automatikusan átszálltak az őt foglalkoztató cégre.
2. A ***„Handabanda”*** együttesen létrehozott műnek minősül, ami körülbelül annyit tesz, hogy a ***VMT Rt.*** kezdeményezte és irányította a szoftver fejlesztését, és azt a későbbiekben saját neve alatt hozta forgalomba[3]. (A mű megalkotásában együttműködő szerzők hozzájárulásai pedig oly módon egyesültek, hogy azokat a későbbiekben nem lehet szétválasztani – azaz itt a fejlesztésben nem csak ***Jimmy***, hanem a ***Kapitány***, és a többiek is közreműködtek.)
3. Az Rt. jogátruházás révén vált a szoftver jogosultjává. Ez a szerzés olyan további kérdéseket vet fel, hogy például milyen áron és milyen körülmények között cseréltek gazdát a jogok. Az átruházás ugyanis nem történhetett szóbeli megállapodással, mert az semmis szerződés lenne, hasonlóan ahhoz az esethez, ha az Rt. képviselője, a ***„Kocsma a három rozsdás szöghöz”*** nevű ivóban íratja alá a meglehetősen ittas ***Jimmy*** fiúval a szerződést, melynek keretében az – további 6 felesért – átengedi a kizárólagos jogokat.

Bár ilyen szélsőséges példákkal a való életben nem nagyon találkozunk, mégis egyértelműen kirajzolódik a nagyobb vállalatok törekvése, mellyel a szerzőktől a lehető legalacsonyabb áron próbálják a fejlesztéseket megszerezni.

Vagyoni jogi ipi-apacs

Induljunk ki abból, hogy az eljárás jogszerű volt, tehát a jogokat törvényes keretek közt szerzi meg a jogosult. Lássuk, pontosan mit is engedünk át ilyenkor.
A mű felhasználási területeit két nagy csoportba soroljuk, ezek az anyagi és a nem anyagi felhasználási formák[4].

Anyagi felhasználás	Nem anyagi felhasználás
• Többszörözés: a szoftvert több tucat CD-re átmásoljuk	• Nyilvános előadás tartása: a képernyő képét kivetítőn megjelenítjük egy konferencián
• Terjesztés: a CD-re másolt szoftvert kereskedelmi forgalomba hozzuk	• Nyilvánossághoz közvetítés: a tévé valamelyik műsorában bemutatjuk a program képességeit
• Bemutatás: ilyen például az, ha egy kiállításon mutatjuk be a művet (ez persze szoftverre nem igazán jellemző felhasználási forma)	• Átdolgozás: ilyen például az, ha egy program forráskódját megjegyzésekkel látjuk el, vagy ha frissítést (upgrade) adunk ki hozzá

A fenti felsorolás három, a szoftverekkel kapcsolatban kifejezetten lényeges pontot tartalmaz, nevezetesen a többszörözést, a terjesztést és az átdolgozást. Ezekkel tehát ismerkedjünk meg közelebbről is.

[1] Ebben a részben főleg kereskedelmi forgalomban megjelenő szoftverekről lesz szó, ráadásul többnyire a szerző szemszögéből. A szabad illetve nyílt forráskódú társaikról külön írást szántam.
[2] Ez szükséges kitétel, a jelenlegi szerzői jogi szabályok szerint, ugyanis csak ez a forma mentes az írásbeliség alól. Bővebben a felhasználási szerződésről szóló cikkben írok erről, valamikor jövőre.
[3] Félreértés ne essék, ettől még a VMT Rt. nem lett szerző.
[4] A táblázat többnyire a szoftverek szempontjából releváns jogokat tartalmazza.

A többszörözés

Ha épp nem a szoftverjogról szólna ez a sorozat, hanem valamilyen más szerzői alkotással hozott volna össze minket a véletlen, sok szempontból egyszerűbb lenne a helyzetünk. Létezik ugyanis a „magáncélú másolás" nemes intézménye, amely lehetővé teszi, hogy egyes művekről (például film a tévében) külön engedély nélkül másolatot készítsünk. Azt persze nem mondanám, hogy ezt minden külön díj megfizetése nélkül tesszük, mert ez a megállapítás nem lenne helyénvaló. Az ilyen másolatokért bizony fizetünk.

Ez a díj hanghordozók esetében (például kazetta, CD, DVD) az úgynevezett ***„üreskazetta jogdíj"***, amit eleve minden egyes adat- vagy hanghordozó árába beépítenek. A másik ilyen – talán nem annyira közismert – „védővám" a papír alapú tartalom többszörözést ellensúlyozó ***„reprográfiai jogdíj"***, amit korábban csak a fénymásolókra alkalmaztak, mostanra azonban – az internetes tartalom egyre terjedő többszörözésére reagálva – egyes nyomtatókra is kiterjesztettek. Ezek a pénzek aztán egy precíz szabályozás szerint a szerzők között kerülnek szétosztásra.

Eddig ha az előbb vázolt gondolatmenetemet megpróbáltam egy informatikussal megosztani, az mindig csúnyán kezdett el nézni rám. Hiszen miért is fizessen ő holmi – általa nem is ismert szerzőknek – mikor a megvett CD-re jó eséllyel csak egy adatmentés kerül? Igazságtalannak és méltánytalannak érezte ezt az eljárást, mint ahogy valószínűleg az a kereskedő is háborog, aki csak számlanyomtatásra használja a lézernyomtatóját. Mit is lehet erre válaszolni? Talán azt, hogy ez a kisebb rossz. Gondoljunk csak bele, hová vezetne, ha a magáncélú másolás is engedélyköteles lenne, vagy esetleg teljesen tilos? Lényegesen csökkenne annak a lehetősége, hogy ismeretekhez jussunk vagy juttassunk másokat. A legegyszerűbb tehát az, ha hőbörgés helyett a kasszánál való sorbanállás alatt azzal a jóleső érzéssel készítjük elő a valamivel magasabb összeget, hogy – ha nem is teljesen jószántunkból – egy jó ügyet támogatunk. Érezzük magunkat inkább mecénásnak, mint becsapott, megsarcolt vásárlónak.

A többszörözés szoftver esetében egyetlen „biztonsági másolat" készítése erejéig tekinthető jogszerűnek. Értelemszerűen a biztonsági másolatról nem illik biztonsági másolatot készíteni, „biztos ami biztos" alapon. Külön kérdést vet fel, hogy egy-egy teljes mentés készítése sérti-e ezt az elvet, hiszen itt a többszörözés készítőjének nyilvánvalóan nem áll szándékában, hogy egy operációs rendszerről készítsen másolatokat. Ő csak adatbázisokat akar – a megfelelő környezetbe illesztve – lementeni. Aztán hogy ezt adott esetben majd hogyan fogja megítélni egy bíróság egy eljárás során, nos azt nem tudhatjuk. Meglátásom szerint a jogsértés ilyenkor annyira csekély mértékű, amellyel kapcsolatban sem társadalomra való veszélyességet, sem pedig ténylegesen a szerzőnél jelentkező kárt nem lehet kimutatni.

A többszörözés jogának megsértése olyankor már sokkal nyilvánvalóbb, amikor elszedve a szomszéd kisfiú kétheti zsebpénzét adunk neki egy másolati példányt a legújabb „csihi-puhi" játékszoftverünkből. Nyilván nincs az a bíróság, amelyik ezek után elhinné, hogy mi a törvény által engedélyezett másolati példányunkat a szomszédban akartuk tartani, mert az egy tucat tizenéves gyerek között nagyobb biztonságban van, mint a szobánkban a polcon.

A terjesztés

Először is tételezzük fel, hogy a mű nem egy vírus, vagyis a jogszerű terjesztésnek efféle akadálya nincs. Terjesztésnek minősül a mű eredeti példányának (szoftver esetében ez kevéssé valószínű) vagy többszörözött példányának (jellegzetes terjesztési forma) hozzáférhetővé tétele a nyilvánosság számára, amely történhet kereskedelmi forgalomba hozatallal, vagy forgalomba hozatalra való felkínálással.

Egy számítógépes program törvény által nevesítve átruházással, bérbeadással valamint az ország területére forgalomba hozatal céljából való behozatallal terjeszthető. Ez konkrétan elidegenítést, ajándékozást, bérbeadást, kölcsönzést valamint importot jelenthet.

A magyar szabályozás az általános rendelkezések között nevesíti a jogkimerülés feltételét, mely szerint, ha a műpéldányt a jogosult vagy az ő kifejezett hozzájárulásával más adásvétellel vagy a tulajdonjog más módon történő átruházásával forgalomba hozta, a terjesztés joga az így forgalomba hozott műpéldány tekintetében – a bérbeadás[5], a haszonkölcsön és a behozatal jogának kivételével – a továbbiakban nem gyakorolható. Könyvek esetében például ez teremti meg az antikváriumok létjogosultságát. Szoftverek vonatkozásában ilyesmit nem találunk... Legalábbis még nem. A hiány oka talán mindössze annyi, hogy eleddig senki sem keresete meg a bíróságot azzal, hogy szeretné már megunt szoftverét továbbértékesíteni, és ebben az arra jogosult őt megakadályozza. Talán feltűnne a bíróságnak, hogy az ilyen jellegű magatartás azon túl, hogy a szabad piacnak árt, erősen emlékeztet a joggal való visszaélésre is...

Tapasztalatból tudhatjuk, hogy kereskedelmi forgalomba kerülő szoftverek licencei vagy teljes mértékben kizárják a továbbértékesítést, vagy soha megadásra nem kerülő engedélyhez kötik azt.

Hasonló nehézségekkel szembesülhet az, aki cégek felszámolásával foglalkozik, és a gépállománnyal megszerzi az azon futó szoftverekhez kapcsolódó jogosultságot is. Legalábbis, azt hiszi, hogy megszerezte. Hazánkban tudtommal ezen a téren nincs sem precedens értékű döntés, sem ehhez kapcsolódó per folyamatban. A németek egy lépéssel előttünk járnak, egy ottani bíróság már kimondta, hogy egy szoftver felhasználásának „továbbadását" ilyen módon megakadályozni nem lehet. Hozzá tartozik a történethez, hogy a program az adott esetben is „tucatprogram" volt, nem pedig egyedi fejlesztés.

A következő számban mesélek még egy keveset a többszörözésről, majd az átdolgozásról, amely szoftverek esetében szintén speciálisnak mondható.

Addig pedig Boldog Mikulást!

Dr. Dudás Ágnes

[5] Faludi Gábor meglátása szerint a másolatra vonatkozó határozatlan időre szóló bérleti jog kikötésével is megállapítható helyes jogértelmezés mellett a regionális jogkimerülés.