Két számmal ezelőtt írtam a szotar.kiskapu.hu új bővítéseiről. Szinte szóról szóra megismételhetem magam: „Igaz, még mindig csecsemőkorát éli, de a tartalma folyamatosan bővül, és igyekszünk a jelzett hiányosságokat is javítani benne”. Igyekeztünk lehetővé tenni mindenki számára, hogy a hiányzó szavakat könnyen el tudja küldeni, illetve új szerkezetet adtunk a letölthető változatnak, remélem, így könnyebben használható. Természetesen a szótár hatékonysága nagymértékben azon múlik, hogy milyen szószedettel dolgozik. Nagyon sok olyan szóval kell megküzdenünk, amelyre még nincs jó magyar fordítás, és folyamatosan keresnünk kell a határvonalat: mit tudunk lefordítani és mit nem. Hiába születik több fordítás egy angol szóra, ha azt a szakma nem képes (vagy nem akarja használni). Az egyik olvasó például az alábbiakat írta:

Szerény véleményem szerint nem kellene a magyarítást túlságosan erőltetni, ha az az érthetőség rovására meg. Például cluster → géptelep. Többször neki kellett ugranom a mondatnak, mire megértettem, mi is lehet az a géptelep.

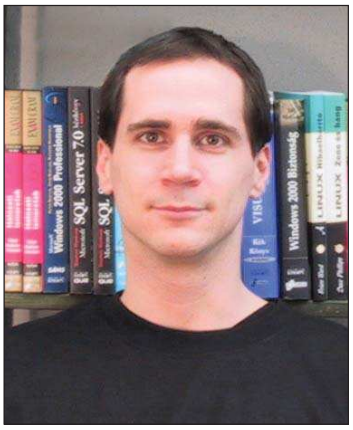
Nem mernék nyíltan állást foglalni, hogy ki melyiket érti jobban. Tapasztalatom szerint ugyanis az emberek többsége (ideértve számos rendszergazdát is) könnyedén egy kalap alá

veszi a cluster, array, farm, de még a stack szavakat is. Mondják, mind ugyanazt jelenti: „Sok izé együtt. És különben is, ha szakemberrel beszélek, úgyis angolul használjuk, nem?” De, angolul, és ugyanúgy pontatlanul. Az ember hajlamos azt hinni, hogy az angol nyelvterületeken élő informatikusok pontosan tudják, mi micsoda. Nos, fityfenét.
Azt viszont el kell ismerni, hogy egy-egy cikk fordítása gyakran sokkal nehezebb feladat, mint az eredeti angol nyelvű szöveg megértése. Ez még akkor is így lehet, ha angolul csak keveset értünk. Például a 30. oldalon kezdődő szakcikk, a „Reiser4: hatékonyan gyorstárazó fák tervezése” így, többszöri átolvasás után is – biztos vagyok benne – tele van félreérthető fordításokkal. Ez egyrészt betudható a fordításnak, másrészt más okok is megbújnak a háttérben. Olyan szakmai különbségekre gondolok, mint például a B és B+ fák szerkezete (különböző iskolákban különböző módon tanítják). Röviden összefoglalva: sok munka áll még előttünk, míg kialakítunk egy egységes, rögzített nyelvezetet, és könnyen lehet, hogy ez csak azokon a területeken történhet meg, ahol a szakma már egységes és letisztult!

---

accessibility – kisegítő lehetőségek (lásd ott)
applet – kisalkalmazás
átirányítás (redirecting) – egy címre érkező kérelemnek (akár szolgáltatás, akár adatküldés stb.) önműködően egy másik címre történő küldése.
consistent – következetes, egységes szerkezetű, szerkezeti sérülésektől mentes
constructor – létrehozó
destructor – megszüntető
engine – motor
folyamat – process
forwarding – továbbítás
kisalkalmazás (applet) – olyan önálló programocska, amely képes valamilyen futtatási környezetbe beágyazódni. Például a weboldalakba, vagy a grafikai alkalmazásokba „beágyazódó” programocska ilyen. A fordítás sajnos hosszú, de eddig nem találtunk jobbat.
kisegítő lehetőségek (accessibility) – a felhasználói felület részeinek használatát (megjelenítés, bevitel stb.) segítő kiegészítések, például csökkent képességű (vak, halláskárosult stb.) – felhasználók számára. Sajnos nem találtunk rá jobb fordítást.
konzisztens – lásd: consistent
létrehozó (constructor) – az a függvény, vagy programrész, amely egy változót (objektumot stb.) hoz létre, alapértékeit, kapcsolatait beállítja.
megszüntető (destructor) – az a függvény vagy programrész, amely egy változó (objektum) megszűntetésekor elvégzi a szükséges feladatokat (kapcsolatok, függőségek bontása, területek felszabadítása stb.).
motor – engine
namespace – névtér
névtér (namespace) – olyan terület, rész vagy egység, amelyen belül csak az adott névtéren belül látható változók hozhatók létre.
overloading – túlterhelés (lásd ott)
portable – környezettől függően jelenthet átültethetőt vagy hordozhatót.
process – folyamat
redirecting – átirányítás
remote login – távoli bejelentkezés
távoli bejelentkezés (remote login) – az adott számítógépre nem közvetlenül történő (például nem a géphez kapcsolt első billentyűzet/monitor páros) történő bejelentkezés.
TCO (Total Cost of Ownership) – Teljes Birtoklási Költség: az összes olyan költség, amely egy adott gép (alkatrész, hálózat stb.) megvásárlása és üzemeltetése során felmerül. Ide tartozik például a karbantartási díj, a követési díj, a kötelező szervizek, az alkatrészkopás, az üzemben tartó személyzet költsége, az áramköltség, az elfoglalt terület bérleti díja.
továbbítás (forwarding) – valamilyen (általában egy másik gép által biztosított) szolgáltatás elérhetővé tétele az adott szolgáltatáson belül. Például SSH-val távoli bejelentkezés közben az SSH az X11-csomagokat is továbbíthatja, ezáltal grafikus programokat is futtathatunk.
túlterhelés (overloading) – programozási nyelvekben (például C++) az a képesség, hogy ugyanazon a néven több változó (függvény stb.) is szerepel, és a kódban a használat módjától függően mindig a megfelelő változó (függvény) kerül felhasználásra. Például van két `f` nevű függvényünk, az első egy, a második két éréket vár. Ha a programban egy értéket adunk át `f`-nek, az első függvényt hívjuk meg, de ha kettőt, akkor a második függvényt. Még nem tisztázott a fordítás sorsa, bár szó szerint és értelmileg is helyes, mégis, továbbra is keresünk más fordításokat is. Javaslatként felmerült a ***kiterjesztés*** is.

Beköszöntő



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

Az egyensúly keresése

Megjelent egy hír, miszerint a Microsoftot 512 millió dollár kártérítés megfizetésére ítélték. Amikor ezt a hírt olvastam, ***Shan-Tung Hsu*** fengshui-mester tanításai jutottak eszembe. Amikor Magyarországon járt, előadása felütéseként a mester azt igyekezett megértetni velünk, hogy a fengshui nem arról szól, hogy a szobámat hogyan rendezzem be, hogy hajtsam le a tojodában az ülőkeszkát, vagy hogy ne rakjak tükröt az ajtóval szembe. Ezek csak következmények. Mint amilyen következménye a náthának a rengeteg taknyos zsebkendő. Tanítása szerint az erőket kell megértenünk. Az erők áramlanak, egyensúlyra törekszenek, és folytonosan hatnak egymásra, és ezek a hatások előbb-utóbb körbeérnek.

A fejlődés során a számítástechnika világa egyszer eljutott egy görcsös állapotba. A hatalom néhány óriásvállalat kezében összpontosult, a megoldások drágák, nehezen hozzáférhetőek voltak. Ekkor néhány kishal nagyon ügyesen elcikázott a nagyok mellett, és új lendületet adott – új típusú erőket hozott a piacra. Ilyen kishal volt a Microsoft is. Később az új erő képviselője világcéggé nőtte ki magát, szép lassan ugyanazokat a célokat tűzve maga elé, mint egykor legyőzött elődje. És megszülettek az új kishalak. Ezek a kishalak mára elcikáztak a mai óriások mellett. És utána?

És utána ők is elkezdtek úszni a korábbi kishalak céljaihoz kísértetiesen hasonlító célok felé.

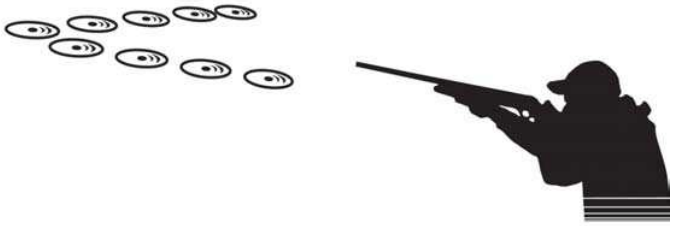
De nézhetjük más szemszögből is.

Az emberek szeretnek harcolni. Nem vallják be, de egyszerűen nem bírják elképzelni a békés egyensúly állapotát. Harcolni kell. Harcolni kell az óriások ellen, ha óriás vagy, a kishalak ellen, harcolni kell a fejőstehénért, aki évente fizeti a jogdíjat, ha fejőstehénnek érzed magad, harcolnod kell a szabadságért, ha szabad vagy, harcolnod kell mások jogaiért, ha jogaid vannak, harcolnod kell a jogdíjért, ha ingyen akarod a világnak adni a terméked, harcolnod kell az ingyenességért. Úgy látszik, ha valaminek értéke van, akkor azért harcolni kell. És, természetesen, ellene is.

Az erők oldaláról nézve ez egyszerű: ha valami kitér az egyensúlyból, a másik oldalt rántja maga felé. Ha valaki világuralkodó akar lenni, szabadságharcosokat követel ki a világtól. Ha valaki fel akarja szabadítani az egész világot, zsarnokok bölcsőjét teremti meg. Fontos tehát, hogy tisztában legyünk az erőkkel. És természetesen a célokkal. Mert a célok is folyamatosan változnak. Az egyik nap még azon dolgozunk, hogy működjön a rendszer, a másikon pedig arra ébredünk, hogy valaki jogdíjat akar beszedni rá. Mint ahogy az üres lemezekre is jogdíjat fizetünk manapság. Ha úgy nézzük, ennek is megvan a maga előnye. Az ember már nyugodtan mondhatja, hogy a lemez csupán házi másolata az egykor megvásárolt hanglemeznek. Még a jogdíjat is megfizettük.

Ebben a nagy kavarodásban az ember csak kapaszkodik, és reméli, hogy a sok őrültnek – akik például azon civakodnak, hogy eredetileg melyikőjük találta fel a mondatvégi pontot (Amerikában erre is jogvédelmet akartak bejegyeztetni) – egymást való perelgetése közben valamelyik hatóság nem dönt úgy, hogy a születése után törvénytelenül sírt fel, ezért pedig súlyos bírságot kell fizetnie. De ebben az esetben a Római Egyház perelné be a hatóságot, hogy jogdíj nélkül alkalmazza az eredendő bűn elméletét.

Programvadászat



Böngészők hada áll rendelkezésre Linux alatt – mostani CD-mellékletünkön a legújabb és „legfontosabb” böngészőket adjuk közre. Ez természetesen mindenkinek más és más: akad, aki a Mozillát, a Netscape-et vagy az Operát kedveli. Akik KDE-felületet használnak, azok nagy része a Konqueror böngészőt részesíti előnyben, ezért mi megkísérlünk mindenkinek a kedvében járni.

Firebird

Mozilla-alapokon nyugvó böngésző, ez azonban tényleg csak egy böngésző felesleges sallangok nélkül, számos más sallanggal viszont kiegészítve, ami segít-

heti a böngészést. Böngészőnek kiváló. A Bongeszok/Mozilla_Firebird könyvtárban találhatnak rá az érdeklődők.

Opera 7.11

Természetesen a legfrissebb megbízható Opera böngésző sem maradhatott ki a sorból – a szokásos formátumokban akadhatunk rá a korongon (rpm, deb, tar.gz). Az Opera még mindig csekély erőforrásigényével tűnik ki a többi

„memóriafaló” böngésző közül. Gyors, megbízható és a legtöbb akadályt, ami az interneten megtalálható, könnyedén veszi. Egyedüli idegesítő vonása az a bizonyos reklámcsík a jobb felső sarokban, ami eltüntethető ugyan, de ezért fizetnünk kell.
Megtalálható a Bongeszok/Opera könyvtárban.

Netscape 7.1

Hosszú fejlesztői munka után végre megjelenhetett a Mozilla 1.4, ebből egyenesen következik, hogy a Netscape is fejlesztett a böngészőjén. Szerencsére a híreszteléssekkel ellentétben nem az 1.2-es Mozilla lett ennek a csomagnak az alapja, hanem a legfrissebb 1.4-es. Korongunkon a hálózati és a teljes telepítőkészlet is megtalálható a Bongeszok/Netscape könyvtárban.

Mozilla 1.4

Mint fentebb már írtam, a Mozilla 1.4-es változata hosszú fejlődés után jelent meg. Nagyszámú újdonságot és hibajavítást követően 2003. június 30-án adták ki a fejlesztők. Tapasztalataim szerint gyorsabb, mint az elődei voltak. Megta-

lálható a Bongeszok/Mozilla könyvtárban, mind telepíthető, mind forráskód formában.

Mozilla 1.5a

A böngészők legfiatalabbja, a „felnövekvő nemzedék”, a következő fejlesztői Mozilla-változat. Csak ínyenceknek, fejlesztőknek vagy elszántaknak ajánlott. A Bongeszok/Mozilla könyvtárban helyeztük el.

OpenOffice.org 1.1RC3

Az OpenOffice.org-csapat is gőzerővel fejleszti csomagját – meg is lett az eredménye a rengeteg befektetett munkának! Gyorsabb és kevésbé nyűgös programot kaptunk, bár ez még csak az RC3-as kiadás, de már nagyon közel vagyunk a célhoz. Sokkal jobbak lettek a Microsoft-formátumok szűrői, valamint PDF, SWF, DocBook formátumok beolvasására is képes.
A CD-n az OpenOffice.org könyvtárban helyeztük el, vállalkozó és a programozáshoz kedvet érző olvasóinknak forráskódban is közreadjuk.

PostgreSQL

Az általam igen kedvelt PostgreSQL legfrissebb, 2003. július 29-én megjelent 7.3.4-es változata is helyet kapott. Ennek a forráskódját adjuk közre, hogy mindenki a saját szájízének megfelelően fordíthassa le. Minden PostgreSQL-használónak javasolt a frissítés, különös tekintettel a 7.3.3-as változatot futtatóknak. A 7.3-as bármely alváltozatról frissítőknek nem kell a dump–reload párost végrehajtania.
Megtalálható még a 7.4beta1-es változat forráskódja is, de mivel ez nem megbízható változat, csak „guruknak” ajánlott. A korong PostgreSQL könyvtárában találhatóak.

2.6.0-test3

A címből is kiderül, hogy a 2.6-os sorozatú rendszermag teszt3-as változata került fel a korongra. Seregnyi hibajavítás után egy még mindig csak próbálgatásra és ismerkedésre alkalmas rendszermagot kapunk, erre a feladatra azonban kiválóan megfelel, így amikor a végleges 2.6.0-változat megjelenik, nem fog meglepetésként hatni ránk a sok újdonság.

Magazin

A szokásos, magazinhoz tartozó anyagok kaptak itt helyet, minden cikknél a cím mellett kis CD-vel jelezzük, ha a korongon hozzátartozó anyag található. E havi anyagaink közül a wxWindows-hoz tartozó csomagoknak és egyéb hasznos dolgoknak jut a legtöbb hely.

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

Áldásuk rá

Az IEEE után a ZigBee Alliance – hálózati termékeket gyártó cégek nonprofit egyesülése – is szabványként fogadta el a 802.15.4-es jelzésű szabálygyűjteményt.

A 802.15.4 megbízható, biztonságos és kis energiafelhasználással végzett távoli felügyeletet és alkalmazásvezérlést tesz lehetővé vezeték nélküli hálózatokon. A szabvány az ISO OSI modell alsó két rétegét (fizikai és közeghozzáférés) fedi le, a megbízható csomagtovábbítást visszaigazolásokkal, hibajavítással, fontossági sorrend felállításával biztosítja, illetve a frekvenciaváltások levezérlésére is alkalmas is. A 2,4 GHz-es és a 868/915 MHz-es frekvenciákat egyaránt lefedi, így világszerte számos távirányítási feladathoz lehet majd alkalmazni, nemcsak számítógépes környezetben, de például központilag vezérelhető háztartások kiépítéséhez is.

➲ http://www.zigbee.org

Legénykedő SCO Group

Sokan várják érdeklődéssel a SCO Grup által az IBM ellen indított peres eljárás végét. Amennyiben a bíróság igazat ad a SCO Groupnak, és megállapítja, hogy az azóta jogi ellentámadást is indított IBM jogtalanul épített be unixos megoldásokat a Linuxba, akkor az összes üzleti célból fenntartott Linux-kiszolgálón módosításokat kell majd végrehajtani. Meg kell ugyanis keresni minden olyan kódrészletet, amely a SCO szellemi tulajdonát képezi, és a linuxos közösség által készítettre le kell cserélni – elvileg.

Ezt kézzel nyilván senki sem szeretné végigcsinálni – legalábbis az Aduva Inc. részéről így gondolják. A cég OnStage és SoundCheck néven két terméket forgalmaz, mindkettő a linuxos kiszolgálók felügyeletének megkönnyítését szolgálja. Az utóbbi ingyenesen is elérhető, és amennyiben a bírósági eljárás a világ nagyobbik részére nézve kellemetlenül ér véget, akkor a cég egy külön változatot fog készíteni belőle, amely a kisebb-nagyobb hibák mellett a SCO Group által jegyzett kódrészletek azonosítására is alkalmas lesz.

Természetesen, ha valaki nem akar ilyesmivel vacakolni, élhet a SCO Group gáláns ajánlatával is, és megvásárolhatja a kérdéses kódrészletek használatának a jogát. A SCO Intellectual Property License for Linux a unixos kódok bináris formában való használatára jogosít fel, ahogy azok a Linux-terjesztésekben megtalálhatók. Így a 2.4-es vagy 2.5-ös változatú rendszermagot használók, miközben a Linux-terjesztésekre vonatkozó GPL szerződés feltételeit is maradéktalanul betartják, megszabadulhatnak attól a nyomasztó gondolattól, hogy esetleg jogtalanul futtatják gépükön a SCO által birtokolt kódrészleteket. A SCO – október 15-ig bevezető áron – mindössze 699 dollárt kér a használati jog megadásáért; ez az ár természetesen egyetlen processzorra értendő. A több processzorral felszerelt, beágyazott vagy asztali rendszerekhez a SCO hamarosan további megoldásokat is ki fog dolgozni.

➲ http://www.aduva.com/soundcheck
➲ http://www.sco.com/scosource

Dugd össze, és…

A Canon i560 jelzésű nyomtatója az első olyan tintasugaras nyomtató, amely a PictBridge megoldás révén – a gyártótól függetlenül – közvetlenül a fényképezőgépről vagy a videokameráról is képes fogadni a kinyomtatandó képeket – amennyiben a képrögzítő eszköz erre alkalmas. Az i550-es modell utódaként

megjelent i560 nemcsak kényelmes, számítógép nélküli használatot nyújt, de a sebessége is kiváló: fekete-fehér oldalból 22, színesből pedig 15 darabot nyomtat ki egy perc alatt, míg legnagyobb felbontása 4800×1200 dpi.

A PictBridge megoldást támogató készülékek képesek a korábbi Bubble Jet Direct megoldás szerinti eszközökkel is együttműködni. A PictBridge-t támogató fényképezőgépek és nyomtatók egyetlen USB-kábellel összekapcsolhatók, majd a segítségükkel margó nélkül nyomtathatók ki a megszokott méretű fényképek. Az i560 nyomtató az Exif Print megoldást is támogatja, amely a fényképezőgép által rögzített kiegészítő adatok segítségével a fényképek élethűbb reprodukcióját teszi lehetővé. Az i560 ajánlott végfelhasználói ára 129 dollár lesz.

➲ http://www.canon.com

Itt van az egeremben

Az IOGEAR Memory Mouse néven bemutatta az első olyan kisméretű, sokat utazó felhasználóknak szánt egeret, amely 32 MB flashmemóriát is tartalmaz. Ha valaki a jövőben nem szeretne hordozható gépet hurcolni magával, és például USB-s flash-meghajtóra sem futja neki, akkor – mi sem egyszerűbb – saját egerének bendőjében hurcolhatja magával bemutatóját, dokumentumait, vagy éppen kedvenc fényképeit, zeneszámait. A Memory Mouse 800 dpi-s felbontással dolgozik, így nagy pontosságot kívánó munkákhoz is használható, illetve szűk helyen a kisebb kézmozdulatok révén kényelmesebb egerészést tesz lehetővé. A géphez USB-felületen csatlakozik, kábele behúzható, így szállítása is egyszerűbb, mint a hagyományos, a hordozható gép táskájában általában kellemetlen gubancolódást okozó vezetékkel ellátott példányoké. Az apróság ára 49 dollár.

➲ http://www.iogear.com

A papír a lényeg

Az IBM és a SuSE közös bejelentése szerint a Linux a két cég munkájának eredményeképpen megkapta első biztonsági minősítését. A SuSE Linux Enterprise Server 8 egy IBM eServer xSeries gépen futva a Common Criteria feltételrendszere szerinti EAL2-es minősítést nyerte el. A munka tovább folytatódik, a SuSE még az idén szeretné megszerezni az EAL3+ szintű minősítést, illetve az Egyesült Államok védelmi minisztériumának kereskedelmi termékekre vonatkozó Common Operating Environment tanúsítványát, így igazolva, hogy operációs rendszere a szolgáltatásokra és a biztonságra vonatkozó kívánalmaknak egyaránt megfelel. A Common Criteria elismerés elnyerése fontos üzenet a szigorú követelményeket támasztó kormányzati szervek vagy vállalatok számára, illetve a Linux fejlődésének szempontjából is lényeges előrelépést jelent, hiszen az év végére – ha minden jól alakul – ilyen módon minősítésekkel teleaggatott SuSE-terjesztés egészen új, eddig haladásellenes módon kiváró piacokat nyithat meg a SuSE, és így az egész linuxos tábor előtt.

➲ http://www.ibm.com/linux
➲ http://www.suse.com

CD-író és kártyaolvasó egyben

Az I/OMagic bejelentette az első olyan belső CD-újraíró meghajtót, amely egyben hétféle memóriakártya kezelésére képes kártyaolvasóként is használható.

A meghajtó 52× sebességgel képes a CD-k írására, újraírást 32× sebességgel végez, és ha az alaplap egyik USB-csatlakozóját hajlandóak vagyunk feláldozni, akkor az IBM Micro Drive meghajtók és a Compact Flash, Multimedia Card, Secure Digital, Memory Stick, Memory Stick PRO és Smart Media kártyák kezelését is helytakarékos módon oldhatjuk meg vele. A cég hasonló meghajtókat eddig is kínált, ám csak külső kivitelben. Az új egység szállítása szeptemberben kezdődik meg, bevezető ára mindössze 79 dollár lesz.
➲ http://www.iomagic.com

Ők ellopják, mi fizetjük

A Kensington új biztonsági termékeket mutatott be, ezek egyike a MicroSaver Guaranteed Notebook Replacement, amely valójában már nemcsak egy termék, hanem egyfajta szolgáltatás is.

A gyártó vállalja, hogyha valakinek ellopják a hordozható számítógépét, holott az a Kensington fent nevezett termékével volt lezárva, akkor a vásárlástól számított egy éven belül történő lopásnál esetenként legfeljebb 1500 dollár, személyenként évi 10 000 dollár erejéig kártérítést fizet a károsultnak, legalább részben pótolva a gép árát. Az új MicroSaver kevlár borítást kapott, belsejében rozsdamentes acélsodrony található, a cég állítása szerint a huzal 40 százalékkal erősebb, mint a korábbi példányok. Esetében is érvényes a Kensington által nyújtott élettartam-garancia. A termék ára 55 dollár. A visszatérítés igénybe vételéhez a vásárlás tényét a vásárlástól számított 30 napon belül be kell jegyeztetni a gyártónál.
➲ http://www.microsaver.com

Linuxos cégek támogatási szövetsége

A TSANeten – Technical Support Alliance Network – az iparág legnagyobb gyártóktól független támogatási szövetségén belül új közösség jött létre: a Linux operációs rendszerrel foglalkozók társulása. Az alapító tagok: a BEA, a Dell, az EMC, a HP, a Network Appliance, a Novell, a SuSE, a Unisys, a Veritas és a VMware, de hamarosan további jelentkezőkre is számítani lehet. A közösség törekvése az, hogy akkor is hatékony támogatást tudjon nyújtani a vásárlóknak, ha azok több gyártótól származó elemekből – amelyek alatt eszközöket, operációs rendszert és alkalmazásokat egyaránt lehet érteni – összeállított rendszert üzemeltetnek. A cél nem kifejezetten az, hogy a vállalatok megosszák a tudásukat egymással, hanem az, hogy elkerüljék az ügyfél „egymásnak való lepasszolását", aki rossz esetben a különféle cégek között őrlődve sehonnan nem kap érdemi segítséget.
A gyorsabbá és pontosabbá váló támogatás révén a Linux alapú rendszerek a komoly elvárásokat támasztó vállalatokhoz, küldetéskritikus környezetekbe is eladhatóvá válnak.
A linuxos közösség a negyedik szerveződés, amely a TSANet keretein belül jön létre, a korábbi három támogatási társulás Microsoft Datacenter, Storage Networking Industry Association és Mission-Critical Customer néven állt össze.
➲ http://www.tsanet.org

Érdemes megjegyezni: CRESERC

A Hitachi, a Mitsubishi és az NTT közös munkájának eredményeként napvilágot látott a CRESERC nevű titkosítási megoldás, amely elliptikus görbékkel dolgozó algoritmuson alapulva valósít meg nyilvános kulcsú titkosítási rendszert. A CRESERC kidolgozásakor – mint a hasonló megoldásoknál általában – finom egyensúlyt kellett találni a biztonság és a hatékonyság között, de a kutatók a saját állításuk szerint sikerrel jártak.
A CRESERC kisméretű kulcsokat használ, mégis magas fokú biztonságot garantál, így a nyilvános kulcsú megoldások új, az RSA algoritmus leváltására is alkalmas nemzedékét képviselheti. Ebben fontos szerepe lehet annak, ha a különféle szervezetek szabványként is elfogadják, márpedig erre jó esély mutatkozik, hiszen az érintett cégeknek több terméke szintén kapott már ilyen kitüntető címet.

Tungsten T2

A Palm megújította csúcsmodelljét: a Tungsten T utódaként megjelentette a Tungsten T2 kézigépet. Az új Tungsten 32 MB memóriát kapott, pontosan kétszer annyit, mint elődje, operációs rendszere a legújabb Palm OS, és kijelzője is élesebb, szebb képet ad. A korábbi változathoz hasonlóan a gép képes a Bluetooth-kapcsolatok kezelésére és hangrögzítésre, rendelkezik elektronikus levelezésre, böngészésre, SMS-küldésre használható ügyfélprogrammal, illetve MP3-fájlok vagy mozgóképek lejátszására alkalmas.

A Tungsten T2 ára 399 dollár. Megjelenésével egy időben – kellemes mellékhatásként – a Palm csökkentette az m130 és az m515 készülékek árát.
➲ http://www.palm.com

Tally és Genicom – egyesülés

Ha valakinek kellemes emlékei kötődnek a Tally névhez, vigyázó szemeit ezentúl a TallyGenicom névre vesse. Az új név rég-új céget takar, amely a Genicom LP és a Tally AG összeolvadásával jött létre. A TallyGenicom a maga 200 millió dolláros éves bevételével az egyik legnagyobb lesz azok között a vállalatok között, amelyek kizárólag nyomtatókkal, nyomtatási kellékekkel és a kapcsolódó szolgáltatásokkal foglalkoznak. A változásból a cég meglévő ügyfelei, vásárlói elvileg semmit nem fognak érezni, legalábbis az árak emelkedésétől vagy a szolgáltatási feltételek megváltozásától nem kell tartaniuk.
➲ http://www.tallygenicom.com

DVD-lejátszó a Lindows.com-tól

A Lindows.com jóvoltából megjelent az első kereskedelmi DVD-lejátszó program Linux alá.
A Lindows.com jogtiszta módon tett szert a DVD-k tartalmának dekódolásához szükséges kódrészletekre, így a lejátszó nem meglepő módon fizetős: 40 dollárba kerül; igaz, a hasonló célt szolgáló alkalmazások árát tekintve ez kedvezőnek is nevezhető.
A program nemcsak a DVD-kkel, de az AVI, MOV, WMV és MP3 formátumokkal is elboldogul. Többek között a Lindows.com oldalán vásárolható meg.
➲ http://www.lindows.com/dvd

Quake-csata útközben

A jövő év a mobil grafika lendületes térnyerését fogja hozni vélik az iparág szereplői. A Khronos Group, amely a mobil eszközökre szánt OpenGL Embedded Systems szabványt állítja össze, elkészült az előírás gyűjtemény 1.0-s változatával. Eközben a Microsoft sem tétlenkedik, hamarosan napvilágot lát Direct3D Mobile API-ja, illetve a területen fontos szerepet betöltő Palm is saját rendszert készít, amely – szerencsére – fő vonalaiban az OpenGL ES szemléletét követi.

A kialakuló versengéshez hasonló küzdelmet láthattunk néhány éve a PC-s grafikai API-k között, ennek nyertese – legalábbis a játékprogramokat tekintve – a Microsoft erejének köszönhetően a DirectX lett. A mobil eszközök piacán egyelőre jóval több befolyásos szereplő – mobiltelefon- és kéziszámítógép-gyártó – mozog, mint az asztali számítógépeken, így a verseny kimenetelét egyelőre kár megjósolgatni.

A nagyteljesítményű mobilmegjelenítés nemcsak a programok, de az eszközök oldaláról is nagy kihívást jelent. A vezérlőlapkáknak az akkumulátoros üzemmód miatt csak nagyon kevés energiát szabad fogyasztaniuk, ugyanakkor bonyolult átalakítások, lebegőpontos számítások elvégzését kell lehetővé tenniük. A nagyobb lapkagyártók, mint az ATi Technologies, az Intel, a Motorola vagy a Texas Instruments már az új piacra fenik a fogukat, nem különben az nVidia, amely hetvenmillió dollárért vásárolta meg a MediaQ nevű, világcégek beszállítójaként pontosan a mobil grafikai eszközök piacára fejlesztő vállalkozást. Az állások elfoglalása hamarosan befejeződik, a háború nem sokára indul. Nekünk csak annyit dolgunk lesz, hogy a lehető leggyakrabban cseréljük le a mobiltelefonunkat.

Felpörgetve

A Hitachi a világon elsőként kínál 7200-as fordulaton működő 2,5"-os merevlemezt. Az IBM-es időkből ismert Travelstar sorozat E7K60 jelzésű tagját a gyártó elsősorban kisméretű, állványra szerelt kiszolgálókba ajánlja, mivel az asztali meghajtókét megközelítő teljesítményének és kis hőtermelésének előnyei ilyen környezetben mutatkoznak meg leginkább. Az új Travelstar 60 GB kapacitású, belsejében két darab üveg alapú korong pörög, az adatok írását–olvasását négy darab fej végzi, súlya 115 gramm.

➲ http://www.hgst.com

A vonalkód visszavág

Japánban megjelentek az első olyan mobiltelefonok, amelyek képesek a vonalkódok olvasására. Mire jó egy ilyen készülék? A Java alapú alkalmazásoknak és a telefonok egyre bővülő memóriájának köszönhetően – a kéziszámítógépekhez hasonlóan – hamarosan üzleti alkalmazásokban is használhatók lesznek, ám egyelőre sokkal köznapibb célt szolgálnak. Ha valaki rendelni szeretne egy katalógusból, azt a távol-keleti országban már mobiltelefonon is megteheti, a vonalkódolvasó a cikkszámok hosszas bepötyögésétől kíméli meg a kedves vásárlót. Miután az érdeklődő a katalógusból kikereste a kívánt árut, egyszerűen az alatta található vonalkódra irányítja mobiltelefonja figyelmét, amely beolvassa a megfelelő számsort, és már el is lehet küldeni a rendelést a megfelelő számra.

A jelenleg még fennálló műszaki akadályok elhárítása után a telefonok arra is alkalmasak lesznek, hogy vonalkódokról például webcímet, telefonszámot olvassanak be nagy sebességgel, amelyet a felhasználó így egyetlen mozdulattal meglátogathat vagy felhívhat. A kétdimenziós kódok beolvasása sem lehetetlen, ezekkel viszonylag nagy adatmennyiséget is kényelmesen lehet majd bevinni a telefonokba.

Viszlát, Ximian!

A Novell – nem nyilvános feltételek mellett – felvásárolta az eddig magánkézben lévő Ximiant. A Linux alá a vállalatok számára is fontos alkalmazásokat fejlesztő Ximian megszerzése révén a NetWare visszaszorulása miatt kicsit szerepvesztetten ténfergő, egyre inkább a tanácsadás és a szolgáltatások felé forduló Novell átfogóbb csoportmunka-megoldásokat kínálhat ügyfeleinek.

A Ximian alapítói, ***Miguel de Icaza*** és ***Nat Friedman*** a továbbiakban a Novell csapatában folytatja munkáját. A Gnome és a Mono projekt mögött külön alapítvány, nagy cégek és önkéntes fejlesztők százai állnak, így ezek sorsát a felvásárlás ténye remélhetőleg nem befolyásolja hátrányosan.

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

Rendszermag-fejlesztési hírek

Egyes fejlesztők – például Alan Cox – azt állítják, hogy a szolgáltatásbővítés befagyasztása tulajdonképpen már nincs érvényben.

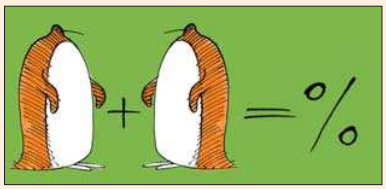
A 2.5-ös fában megváltozott a megszakításkérés-kezelők visszatérési értéke, hogy jobban tudjanak kezelni bizonyos, külső eszközökkel kapcsolatos hibákat. Ennek az lett az egyik következménye, hogy sok-sok vezérlőprogram használhatatlanná vált, és ki kellett javítani a forráskódjukat. A hivatalos rendszermagforrás részét képező vezérlőprogramok szemszögéből ez a változás visszatérő, nemkívánatos napi teendőnek tekinthető, ami azonban véghezvihető. Ezzel szemben számtalan külső vezérlőprogram csak akkor lesz kijavítva, amikor valaki észreveszi, hogy már nem működnek, sőt a jogdíjas vezérlőprogramok esetében még inkább elhúzódhat a javítás.
Bartlomiej Zolnierkiewicz az IDE-karbantartás hivatalos felelőse. *Andre Hedrick* – legalábbis pillanatnyilag – félreállt, hogy a soros ATA- (SATA) illesztéssel és a gyártók lapkakészleteivel kapcsolatos kérdésekre összpontosítson. Továbbra is *Alan Cox* a hivatalos összekötő kapocs Linusszal az összes IDE-javítókészlet tekintetében. Andre úgy döntött, hogy visszamenőlegesen az összes olyan munkáját kiadja, amellyel hozzájárult a rendszermag fejlesztéséhez, mégpedig kettős felhasználási engedéllyel, nem csak a GNU általános közreadási feltételek szerint. A másik ilyen *Lawrence E. Rosen* nyílt szoftverfelhasználási engedélye.
Benjamin Herrenschmidt átvette a Radeon Framebuffer kódot, amikor *Ani Joshi*, a kód hivatalos karbantartója abbahagyta a javítókészletek alkalmazását, és az üzenetekre sem volt hajlandó válaszolni. Ani eltűnése után Benjamin összegyűjtötte a fellelhető különböző javítókészleteket, és kiadta az új RadeonFB-frissítést.
2003. május elején még mindig sok tennivaló akadt a vezérlőprogrammal, és Benjamin úgy tervezi, hogy a 2.4-es fa jelenlegi alapkódjával folytatja a munkát, és a szolgáltatásbővítés lezárása ellenére megpróbálja bejuttatni a program teljesen átírt változatát a 2.5-ös fába. Egyes fejlesztők – például Alan Cox – azt állítják, hogy a szolgáltatásbővítés befagyasztása tulajdonképpen már nincs érvényben. Ha ez tényleg így van, akkor a 2.6-os (vagy 3.0-s) változat előkészítése jegyében elrendelt, a 2.5-ös változat állandósítására tett első kísérlet meghiúsult.
Az I/O-vezérlőfüggvények (`ioctls`) letűnőben vannak, amióta a SysFS a 2.5-ös rendszermag tervezési megbeszélései során megjelent. Azelőtt sokan kirohantak az ellen, hogy a vezérlőprogramok `ioctls` függvényeket alkalmazzanak. Nem pusztán rengeteg volt belőlük, de egyszerűen nem is lehetett tudni, hogy pontosan hány ilyen függvény létezik, és így megfelelő leírást sem lehetett hozzájuk készíteni. Habár még mindig számos olyan akad, amitől meg kell szabadulni, egyre több jelenik meg az ésszerűbb SysFS felületen. Továbbá az általános tisztogatás jegyében eltávolítják azokat, amelyeket soha senki nem használt, például a `SCSI_IOCTL_SYNC` és a `SCSI_IOCTL_BENCHMARK_COMMAND` függvényt. Úgy tűnik, hamarosan a SysFS lesz az elsődleges illesztési felület a felhasználók és a rendszermag között, felváltva az `ioctls`-t és az engedetlen ProcFS fájlrendszert.
Greg Kroah-Hartman már 2003 eleje óta a `devfs` (és a /dev) lecserélésén munkálkodott, április közepén végül kiadta az `udev` első változatát, ami *Dan Stekloff* tervei és ötletei alapján készült. A */dev* könyvtár mindig is rém rendetlen volt, százával hemzsegtek benne a szükségtelen fájlok.

Zack Brown

Linux Journal, 2003. augusztus, 112. szám

Linux-index

1. A 2002 végén eladott kiszolgálói operációs rendszerek ennyi százaléka volt Linux: **15–20**
2. 2006-ra vagy 2007-re az eladott új kiszolgálói operációs rendszerek ekkora százaléka lesz Lintel (Linux on Intel): **45**
3. Az informatika területén dolgozók alapfizetésének növekedési aránya jelenleg: **5%**



4. Ennyi SCO Unix-kiszolgálót váltanak fel Linux-kiszolgálóval az új-zélandi Farmlands mezőgazdasági kiskereskedelmi üzletlánc kirendeltségeinél: **28**
5. Ennyi gazdálkodót szolgál ki a Farmlands: **15 000**
6. Ennyi Linux-kiszolgálót adtak el Ázsia csendes-óceáni térségében 2002-ben: **18 000**
7. A fenti kiszolgálók eladásából származó bevétel dollárban: **58 millió**
8. Várhatóan ennyi Linux-kiszolgálót adnak majd el Ázsia csendes-óceáni térségében 2007-ben: **47 000**
9. A fenti kiszolgálók eladásából származó bevétel dollárban: **146 millió**
10. Legkevesebb ennyi számítástechnikai csomóponton fut majd Linux az IBM új Blue Gene rendszerében: **65 000**
11. Ennyi CPU van egyetlen Blue Gene lapkán: **32**
12. Ennyi 32-bites CPU lapka van egyetlen Blue Gene-csomópontban: **64**
13. Ennyi csomópont fér egy Blue Gene-keretbe: **8**
14. Ennyi keret szükséges ahhoz, hogy elérjék az 1 PFLOP teljesítményt (1 petaflop = egymilliószor egymilliárd matematikai műveletet másodpercenként): **64**
15. Nagyságrendileg ennyi millió dollárt különítettek el a Blue Gene fejlesztésére a 2000-ben tett bejelentés szerint: **100**

Források
1–3.: Meta Group, Inc.
4–5.: New Zealand Herald
6–9.: Gartner Group
10–15.: CNet

Linux Journal, 2003. 112. szám

Honi internet-saga

A Sulinet Expressz (SEx) Program alapvető célkitűzése, hogy a pedagógusokat és a tanulókat otthoni használatra számítógéphez segítse.

Az adókedvezményre jogosultak körét és a kedvezmény mértékét egészen pontosan a 2002. XLII., az adókról, járulékokról és egyéb költségvetési befizetésekről szóló törvények módosításáról szóló törvény írja le, amely szerint:

> 11. § (3) A pedagógus, az oktató, a hallgató, a felnőttképzésben résztvevő magánszemély, valamint az a magánszemély, akinek gyermeke nappali rendszerű iskolai oktatásban vagy a felsőoktatásról szóló törvényben felsorolt felsőoktatási intézményben hitelesített iskolai rendszerű első alapképzésben vesz részt, az összevont adóalap adóját csökkentheti az adóévben általa számítógép, számítástechnikai eszköz megszerzésére (vásárlására, bérletére, lízingelésére) fordított összeggel, ha a megszerzés a Foglalkoztatáspolitikai és Munkaügyi Minisztérium, az Informatikai és Hírközlési Minisztérium vagy az Oktatási Minisztérium által kiírt pályázat keretében történik. Az adókedvezmény alapját és összegét a vételárat, a bérleti díjat, illetve a lízingdíjat tartalmazó, a magánszemély nevére kiállított számla bizonyítja.
>
> (4) A magánszemély által e § alapján igénybe vehető adókedvezmény(ek) összege adóévenként nem haladhatja meg a hatvanezer forintot."

Ha valaki számítástechnikai eszközt lízingel vagy bérel, és a lízingdíjat vagy bérleti díjat több adóév során számlázzák, akkor a kedvezmény a jogszabály érvényessége alatt minden egyes adóévben igénybe vehető, feltéve, hogy a többi feltétel is teljesül. Sajnos azonban a többéves áruhitel nem tartozik ebbe a körbe, mivel ebben az esetben a termék vételárát a vásárlás időpontjában számlázzák, vagyis csak egyszeri hatvanezer forint adókedvezmény vehető igénybe. Egyetlen magánszemély több címen is lehet jogosult (például pedagógus és szülő), viszont a kedvezmény együttes összege ekkor sem haladhatja meg a bűvös hatvanezer forintos határt. Amennyiben szülőként kívánjuk igénybe venni az adókedvezményt, akkor egy gyermek után mindkét szülő igénybe veheti a legnagyobb kedvezményt, mivel nálunk jelenleg nincs családi adózás, csupán személyi jövedelemadó. A kedvezmények azonban nem vonhatók össze egyetlen termék megvásárlására, mivel ugyanazt a terméket nem lehet kétfelé számlázni. Ehelyett mindkét szülőnek külön-külön kell megvásárolnia egy-egy számítástechnikai eszközt.

A kedvezmény nem a vásárláskor érvényesíthető, ugyanis akkor természetesen ki kell fizetni a termék vételárát, hanem az adóév végén, az adóbevallás elkészítésekor jelentkezik: a vásárlónak a kedvezmény összegével csökkentett, azaz hatvanezer forinttal kevesebb adót kell befizetnie. Arra azonban nagyon oda kell figyelni, hogyha az egyébként megfizetendő adó a kedvezmény összegénél kevesebb, akkor a különbség elvész, visszatéríteni nem lehet.

Ki lehet igényjogosult?

Az igényjogosultságot nem a vásárlásnál, hanem csak az esetleges adóellenőrzésnél kell igazolni, viszont az igényjogosultságnak a vásárlás időpontjában meg kell lennie. A többéves kedvezmény igénybevételénél (például lízing) a kedvezmény ideje alatt a jogosultságnak végig fenn kell állnia. Ha például a gyermek közben befejezi az iskolát, onnantól kezdve már nem jár a kedvezmény. Mindettől függetlenül a forgalmazó például részletfizetés vagy lízing esetén saját elbírálás alapján kérheti a jogosultságok igazolását szolgáló dokumentumok bemutatását. Ez a szülők esetében az iskolalátogatási bizonyítvány, a pedagógusoknál, oktatóknál pedig a közalkalmazotti jogviszonynak, illetve a munkaviszonynak az adott időpontban való fennállását igazoló okirat.

Hol mit mennyiért?

Eladói oldalról a törvényalkotó a részvételt pályázathoz kötötte, így azon cégek, konzorciumok termékei kerülhetnek a SEx-en belül értékesítésre, akik erre pályáztak és nyertek. A programban megközelítőleg hetven cég vesz részt, a jogosultak az ország számos pontján választhatnak a több mint 250 féle számítógép, hetvenféle program – közöttük természetesen Linux-változatok is, például SuSE Linux –, valamint nyomtatók, kivetítők, digitális kamerák és modemek közül.

Az eddigiek során e területen történt a legtöbb baklövés, sokáig sűrű homály fedte a mit-honnan-mennyiért kérdéskört. A pályázatnyertesekkel megkötött keretszerződés alapján az Oktatási Minisztérium minden értékesítőhely számára SEx-emblémával ellátott, A4-es méretű Sulinet Expressz értékesítőhely tanúsítványt adott ki. A tanúsítványokat az eladótérben ki kell kifüggeszteni, megkönnyítve ezzel a vásárlók számára a döntést, hogy kérdéseikkel, panaszaikkal közvetlenül hova forduljanak. A termékekre öntapadós SEx-matricát és a termék és az értékesítőhálózat azonosítására alkalmas hologramos termékjelölő címkét kell ragasztani.

Néhány szó a kisördögről...

Nyilván sokakban megmozdult a kisördög, ám a pályázat megalkotói a magyar szokásokhoz híven azonnal igyekeztek néhány kiskaput bezárni: a SEx keretében az adókedvezmény igénybevételével vásárolt számítógépet, eszközöket a kedvezményt igénybevevő magánszemélyek a vásárlás évének utolsó napját követő harmadik év december 31-ig nem értékesíthetik, nem adhatják bérbe, vagy másnak térítésért vagy anélkül nem engedhetik át, gazdasági társaságba apport jogcímen nem vihetik be.

➲ http://ado.sulinet.hu/

Nagy Anna (Nagy.Anna@linuxvilag.hu)
Linuxvilág felelős szerkesztője, akit a sors túltengő pedagógiai hajlammal és birkatürelemmel áldott meg. Két kiskorú gyermekével életre szóló programot írt magának.

LME-hírözön

A Linux-felhasználók Magyarországi Egyesületének (LME) életében rendkívül mozgalmasan telt az elmúlt pár hónap.

2003. június 21–22. között *Persányi Miklós* környezetvédelmi miniszter fővédnökségével került megrendezésre a negyedik Ózon fesztivál Budapesten, a városligeti szánkódomb mellett.

A fesztivál célja az volt, hogy megpróbálják összegyűjteni és bemutatni azokat a válaszokat és megoldásokat, amelyeket a civil társadalom ad a létét fenyegető környezeti, gazdasági, szociális és emberi válságokra.
A fesztivál keretén belül beszélgetések, előadások, filmvetítések kaptak helyet, civil szervezetek mutatkozhattak be, és tapasztalatcserére, környezetbarát cégek és módszerek bemutatkozására nyílt lehetőség.
Ezenkívül volt biopiac, és az otthon elkészíthető ételek garmadával (például a házi kefirrel, joghurttal, kenyérrel) ismerkedhettünk meg, a kovászkészítés és a csíráztatás rejtelmei is feltárultak, de életmód-tanácsadáson is részt vehettek az érdeklődők. Bepillantást nyerhettünk a környezetvédelmi jogérvényesítés útvesztőibe, és a független médiák bevonásával (Indymedia), valamint a zöld és kritikai folyóiratokkal (Liget, Ezredvég, Eszmélet, Ökotáj, Kovász, Fedél Nélkül, Gaia), illetve egyéb kisebb újságok, fanzinok, kézírásos újságok bemutatkozásával tapasztalatcserére is sor került.
A „Lehet más a világ!" mottójú eseményen egyesületünk is képviseltette magát: a CD-író projekt számtalan linuxos CD-vel ajándékozta meg az érdeklődőket, illetve *Hazai Géza*, az LME elnökhelyettese *Varga Csaba Sándor*-ral, az FSF.hu alapítvány kuratóriumának elnökével tájékoztató jellegű előadást tartott.

Linux-tábor 2003

Czakó Krisztián (Slapic) jóvoltából harmadik alkalommal került megrendezésre az immáron hagyományosnak mondható Linux-tábor.
A megközelítőleg nyolcvan látogató két turnusban (június 29–július 5., illetve július 6–12.) vendégeskedett Szerencsen, a „Csider-villában", azaz a *Csider Andor* által igazgatott Szerencsi Középiskolai Kollégiumban.
A szállás 2–4 ágyas szobákban, a folyosók végén internetelérással, számítógéptermekkel, a nappalikban televízióval, az alapértelmezett háromszori étkezés lebonyolítására étteremmel várta a tudásra szomjazókat, valamint a Linux iránt kevésbé érdeklődő családtagokat. A napi 4–6 órás oktatás a kollégiumtól 500 méterre lévő gimnázium géptermeiben az alábbi témakörökben zajlott:

1. hét

- **Kezdők számára**
  Általános Linux-alapismeretek
  – Telepítés
  – Linux-rendszer felépítése, TCP/IP
  – Rendszermag
  – Grafikus felület (Gnome, KDE, ablakkezelők)
  – Hang
  – USB használata
  – Belső hálózatok
  – Hálózatbeállítás, névfeloldás
  – Samba
  – Levelezés
  – Nyomtatás (hálózati, cups)
  – Telepítés forráskódból
- **Haladók számára**
  – RAID
  – LVM
  – Programozás: C, PHP
  – Debian-csomagkezelés, -csomaggyártás
  – Biztonság: elmélet, RSBAC, Netfilter, átlátszó proxy, Zorp, VPN
  – Kiszolgálók: Apache, Bind, dhcp, PostgreSQL, cups, Squid, LDAP
  – Lemez nélküli (Diskless) rendszerek
  – Levelezés: Postfix, Qmail, Courier, Squirrelmail
  – Vírus- és levélszemétszűrés
  – Grafikus rendszer távfelügyelete

2. hét

- **Kezdők számára**
  – Linux teljesen kezdőknek
  – Debian-telepítés, -beállítás
  – Fájlrendszerek
  – Kiszolgálók: DNS, DHCP, proxy, Samba, cups
  – Levélkiszolgálók
  – Apache, PHP
  – Hálózatbiztonság, csomagszűrés
  – Grafikus rendszer, hang
- **Haladók számára**
  – Lilo, grub
  – Programfejlesztés
  – Netfilter
  – Policy routing, QoS
  – Kiszolgálók: DHCP, DNS, LDAP (azonosítás), Apache, Samba, Squid

– Levélkiszolgálók: Postfix, Qmail, Vpopmail, Courier (POP3, IMAP)
– Grafikus felület
– Dial-in
– Távfelügyelet

- **Profik számára**
  – Programfejlesztés
  – Debian-csomaggyártás
  – Biztonság elméleti szinten
  – RSBAC
  – Sandbox/jail-készítés
  – Csomagszűrés
  – VPN
  – Applikációs tűzfalak
  – Lemez nélküli rendszerek

Kezdőknek a jelentkezéshez a Linux ismerete nem volt követelmény. Az ismeretanyagok átadását az alábbi oktatók végezték, akiknek ezúton is köszönjük az önzetlen segítséget:

– *Balázs Tibor (Cövek)*
– *Kis Tamás (Dozer)*
– *Szalai Ferenc (Szferi)*
– *Czakó Krisztián (Slapic)*
– *Tomka Gergely (Tomka)*
– *Pásztor György (Pasztor)*
– *Magosányi Árpád (Mag)*
– *Gerendás Zoltán (ZGerendas)*
– *Haluska György (George)*
– *Agócs Péter (Virusbuster Kft.)*
– *Dr. Szentiványi Gábor (ULX Kft.)*
– *Tesch Zoltán (Kancellar.hu)*
– *Bodoky Tamás (Index.hu)*
– *Witch (a „szakértő”)*

A meglehetősen foghíjasra sikeredett programból érdemes kiemelni *Tesch Zoltán* és *Bodoky Tamás* előadását, valamint a vasárnap délelőtti elmélyült beszélgetést, ahol mindenki mindenkivel eszmét cserélhetett az előtérben.
Gyakorlott konferenciaszervezők biztosan egynaposra tervezték volna a rendezvényt, gondoskodtak volna

étkezési lehetőségről, de azt nézve, hogy ez lényegében egyetlen ember (Kovács Krisztián) munkájának az eredménye, csak gratulálni lehet.
A rendezvény alkalmából készített honlap egész évben folyamatosan várja az érdeklődőket a ➲ http://www.hacktivity.hu/ címen. Az eseményen készült videofelvételek az ➲ ftp.eplanet.hu kiszolgálóról a `ftp-hacktivity` felhasználói névvel tölthetők le.
Az eseményt egyesületünk apróbb ajándékokkal támogatta.

I. Hackertalálkozó

„Elmondanám, de nem lehet…” – ez lehetett volna a találkozó mottója, ugyanis a magyar törvények már azt is büntetik, aki a meglévő hibákat pusztán nyilvánosságra hozza.
2003. július 5–6. között került megrendezésre hazánk első „betyár” találkozója Tatabányán, a KPVDSZ Művelődési Házban. Megszokhattuk, hogy az ilyen és ehhez hasonló országos érdeklődésre számot tartó rendezvények rendszerint fővárosunkban kerülnek megrendezésre. *Kovács Krisztián*, a rendezvény fő szervezője ennek ellenére lakhelyét, azon belül is az Újváros szívében található, 1976-ban épült KPVDSZ, azaz a kereskedelem, a vendéglátás, a pénzintézet dolgozóinak szakszervezeti művelődési házát választotta. Az egykori KISKER Kultúrotthon jogutódja a helyi közösségi ház és „befogadó ház” szerepét is ellátja, és ezt saját erőből teszi, tehát nem önkormányzati ház, talán ezért is adott örömmel otthont a hazánkban még szokatlan rendezvénynek.
A rendezvényt hozzávetőlegesen nyolcvan látogató tekintette meg. Mivel a látogatók teljes névtelenséget élveztek, csak tippelni lehet, hogy honnan érkeztek – de a beszélgetésekből, hozzászólásokból arra lehet következtetni, hogy a többség rendszergazda, illetve informatikai, biztonságtechnikai szakemberekből állt.
Előadásokat a következők tartottak:

Sziget fesztivál

2003. július 30. és augusztus 6. között rendezték meg hazánkban a Hajógyári-szigeten Közép-Európa legnagyobb fesztiválját. A Linux-felhasználók Magyarországi Egyesülete az FSF.hu Alapítvánnyal karöltve a „Lehet más a világ” összefogás által létrehozott Zöld Udvar vendégeként első ízben rendezte meg a „Szabad Szoftver Sziget Fesztivált”, amelynek keretében a Sziget vendégeinek nyílt forrású programokat, operációs rendszereket mutattak be, ezeket a látogatók természetesen ki is próbálhatták, előadásokat hallgathattak, valamint az LME CD-író projektjétől és a támogatóktól kapott Szabad Szoftver CD-khez juthattak.
A rendezvény támogatóinak jóvoltából az érdeklődők SuSE és UHU-Linux terjesztésekkel találkozhattak, ezeket gyári CD-ken is megtalálható bővítményekkel vehették

kézbe. A szabad programokkal már jó barátságban lévők a Debiannal és egyéb Linux-változatokkal ismerkedhettek, ezeket a Szigeten a Multimédia Kft. által támogatott CD-író projekt segítségével haza is vihették. Támogatóink közül ki kell még emelnünk a Magyar BSD Egyesületet, az FSN-t, a Linux Support Centert és a Mission Critical Linuxot, valamint a Software Station céget.
Vasárnap *Varga Csaba Sándor*, az FSF.hu alapítvány kuratóriumának elnöke és *Szervác Attila*, az LME és a Szabadon kampány jelenlévő képviselője tartott előadást. Az előadás anyagát médiapartnerünk, az Indymedia rögzítette.
A jelenlévők éppúgy tájékoztatást kaptak a szabadon felhasználható programok társadalmi vonatkozásairól, mint a gazdaságiakról, valamint azokról az előnyökről, amelyek miatt a közigazgatási intézmények, az iskolák, a civil szervezetek, a magyar vállalatok és a magánszemélyek is egyre inkább a nyílt forráskódú megoldások felé fordulnak.
A látogatók megtapasztalhatták, hogy milyen könnyű egy szabad operációs rendszert és a hozzájuk tartozó rengeteg kiváló programot kezelni. A bemutatókat a civil együttműködés jegyében a Semmelweis Orvostudományi Egyetem vezetőképzője támogatta az általuk kölcsönadott 233 MHz-es gépekkel, amelyek megmutatták a szabad programok erejét és kimagasló üzembiztonságát.

Debian X Party

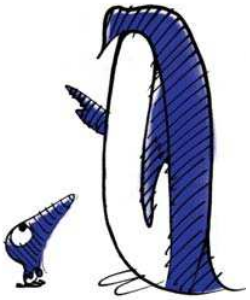
A Linux-felhasználók Magyarországi Egyesületének Debian csoportja 2003. augusztus 15–17-én a Budapesti Műszaki Egyetem Schönherz Zoltán Kollégiumában, az Új Vár Klubban határozta el, hogy a Debian GNU/Linux 10. születésnapja (augusztus 16.) tiszteletére megrendezi a Szabad Szoftver Közösségi találkozóját, amit Debian X Party névre kereszteltek.
A Debian projekt minden idők legnagyobb nyílt fejlesztési modellre épülő szabad programja, amelynek „az univerzális operációs rendszer” megteremtése a célja.
Mára a világ legnagyobb programrendszerévé nőtte ki magát, mert gyors, biztonságos és bármilyen felhasználói igény kielégítésére alkalmas felületet nyújt az otthoni felhasználóknak, cégeknek, irodai alkalmazottaknak és fejlesztőknek egyaránt.
A résztvevők az ünneplés mellett hasznos munkával kívánják tölteni a hétvégét. Erre jó lehetőséget ad a DDTP rendszer (Debian Description Translation Project), amelynek segítségével könnyen fordíthatók azok a rövid, három-négymondatos leírások, amelyek magyarul segítik elő a Debian alapjának telepítését, a beállításokat és a felhasználó igényei szerinti programok telepítését.
Magyarországon akadt már egy példátlanul sikeres akció, amikor az FSF.hu Alapítvány aktivistái által szervezett OpenOffice.org-fordítás alkalmával bebizonyították, hogy egy jól megszervezett akcióval milyen óriási eredményeket lehet elérni. Az LME Debian csoportja hasonló eredményességre törekszik.

V. GNU/Linux szakmai Konferencia

A Linux-felhasználók Magyarországi Egyesülete 2003. november 8-án (szombaton) rendezi meg az V. GNU/Linux szakmai Konferenciát, seregnyi neves előadó és kiállító részvételével a Hotel Novotel Budapest Centrumban (Budapest, VIII. Rákóczi út 43–45.; ➲ http://www.novotel-bud-centrum.hu/indexhu.html).
A tanácskozás területén kiállítás és vásár is lesz, ahol linuxos szakkönyveket, programokat és egyebeket lehet majd megvásárolni. A konferencia fő témája a honosítás és a magyar fejlesztések.
A konferenciával kapcsolatban felmerülő véleményeket, kérdéseket a Konferencia 2003 (konf@lists.linux.hu) listára várjuk. A program reggel 10 órakor kezdődik és 18 óráig tart. Idén a támogatóknak köszönhetően a belépőjegyek ára az alábbiak szerint alakul:

- **Támogatott belépő**
  A támogatott belépő térítésmentes. A támogatott belépővel érkező vendégek a konferenciakiadványon és 1–2 apróságon kívül semmilyen egyéb szolgáltatásban nem részesülnek. Előzetes regisztráció kötelező!
- **Üzleti belépő**
  Az üzleti belépőjegy ára 19 900 + 25% áfa, azaz bruttó 24 875 forint.
  Üzleti belépőt vásárló vendégeink a következő szolgáltatásokban részesülnek:
  – konferenciakiadvány,
  – reklámanyagok,
  – jegyzettömb, reklámtoll,
  – konferencialogós, galléros póló,
  – az előadások bemutatóinak kinyomtatott egybefűzött példánya,
  – névre szóló kitűző,
  – büfé (kávé, üdítő, sütemény),
  – üzleti ebéd a hotel éttermében,
  – a konferencia zárásakor a szervezők nagy értékű ajándékot sorsolnak ki az üzleti belépőtulajdonosok között.
  Az előzetes regisztráció szintén kötelező!
- **VIP-belépő**
  A VIP-belépő térítésmentes, közvetlenül nem igényelhető. Ilyen belépőket az LME díszvendégei kapnak, valamint a támogatók is hozzájuthatnak, és ezeket munkatársaik vagy az általuk meghívott személyek szabadon felhasználhatják.
  A VIP-belépővel érkezők ugyanazokat a szolgáltatásokat élvezhetik, mint az üzleti belépőjegyet vásárló vendégeink.

Gibizer Tibor (gibzo@linuxportal.hu)
Újságíró, immár nyolc éve a Linux elkötelezett híve. Imádja a kutyákat, a kerékpározást és az autós csavargást.



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

A hónap szakmai tanácsai

Kapcsolat időtúllépése

A `telnet` és az SSH-kapcsolatok időtúllépés miatt megszakadnak. A `tcsh` parancsértelmezőt használom, és a `pty`-eszköz, amire bejelentkezem, fel van sorolva a */etc/securetty* állományban. Ez nem az `autologout` hibája. Ha letiltom az `autologout`-ot, akkor is megszakad a kapcsolat, nagyjából egy óra múlva. Amikor ez történik, a felhasználó még mindig bejelentkezettként van felsorolva és a parancsértelmező működik. A folyamatazonosítóját kilőve lehet csak befejezni a futását.
Floyd Miller, floyd@studiodust.org

Nagyon valószínű, hogy a tűzfalban van a hiba. Gyakran megesik címfordító és álcázó tűzfalaknál, hogyha egy összeköttetésen egy bizonyos ideig nincs forgalom, akkor az útválasztó eldobja a kapcsolatot, mert feltételezi, hogy helytelenül lett lezárva. Ennek az az oka, hogy bizonyos ügyfelek nem küldenek megfelelő lezárási kérést, és nem lenne okos dolog fenntartani ezeket a halott kapcsolatokat, mert rendszermag-erőforrásokat kötnek le.
Chad Robinson, crobinson@rfgonline.com

Valószínűleg címfordító átjárón mész át, ami a tétlen TCP-kapcsolatokat egy óra tétlenség után megszakítja. Rendszergazdaként add ki a következő parancsot:

```
echo 600 > /proc/sys/net/ipv4/
➥tcp_keepalive_time
```

Ezután, ha SSH-t használsz, küldj életben tartó TCP-csomagokat, hogy a kapcsolat ne haljon el:

```
ssh -o ´KeepAlive=yes´ célgép
```

Marc Merlin, marc_bts@google.com

Kevesebbet kell gépelned, ha beírod a

```
ProtocolKeepAlives 300
```

sort a *~/.ssh/config* állományba, hogy az SSH minden kapcsolatra ötpercenként életben tartó csomagot küldjön.
Don Marti, dmarti@ssc.com

Támogatás az Intel Videóhoz?

Videokártyám egy beépített Intel 82845G/GL, ami nem hajlandó együttműködni a Linuxszal (Red Hat 8.0). A Linux a telepítés során lekérdezi, de nem indul el a grafikus mód, és a `startx` végzetes hibát ad.
Jafar Borhan, jafar_borhan@yahoo.com

A Google-n keresve találtam egy oldalt, ahol leírják a rendszer beállítását ehhez a videokártyához: ➲ http://www.linuxcare.com/labs/certs/ibm/netvista-m42/rh80-config.epl. Frissítsd a felsorolt csomagokat, majd futtasd az Xconfigurator programot.
Marc Merlin, marc_bts@google.com

Régi rendszermagok kiselejtezése

A Red Hat Network segítségével frissítem a programjaimat. Engedélyeztem az `up2date` programnak, hogy a rendszermagot is frissítse. Viszont a */boot* lemezterület kezd megtelni. Hogyan távolíthatok el néhányat a régi rendszermagok közül? Nem hinném, hogy öt különböző rendszermagra lenne szükségem a */boot* lemezrészen.
Bob Wooden, bobwooden@netwalk.com

Egyszerűen távolítsd el a nem kívánt rendszermagokat. Az `rpm -qa | grep kernel` parancs megadja, hogy mely rendszermagcsomagok vannak telepítve, és az `rpm -e` paranccsal a régiek eltávolíthatók. Azt javaslom, hogy legalább kettőt tarts meg, így ha valami rosszra fordul a jelenlegivel, marad egy tartalék.
Mario Bittencourt, mneto@argo.com.br

Ez nemcsak rendben van, de ez a helyes rendszergazdai magatartás. Csak a hasznos rendszermagokat tartsd meg – általában kettő szükséges: az elsődleges rendszermag és a biztonsági tartalék, ha valami történne az elsődlegessel. Annyi különféle változatot, mint neked van, ritkán kell megtartani, hacsak nem különlegesek az igényeid, például rendszermag-illesztőprogramokat fejlesztesz és próbálsz ki.
Chad Robinson, crobinson@rfgonline.com

USB flashmeghajtó?

Hogyan fűzhetek be egy USB flashmeghajtót? A */proc/bus/usb/devices/* könyvtárban megjelenik a meghajtó. A hardverkereső `hda4`-ként azonosítja `fat32` fájlrendszerrel, de nem tudom befűzni, sem a fájlokat elérni.
Callum Benepe, callumb@yahoo.com

Úgy tűnik, hogy nincs betöltve nálad az `usb-storage` modul, ami viszont szükséges ehhez az eszközhöz. Olvasd el a Linux USB Guide dokumentumot a ➲ http://www.linux-usb.org honlapon. Itt részletesen leírják a megfelelő illesztőprogramok betöltésének és az eszköz befűzésének a módját.
Greg Kroah-Hartman, greg@kroah.com

A Windows eltüntette a LILO menüjét

Számítógépemre Red Hat 7.1 van telepítve, és egy másik lemezrészen Windowst használok. Nemrég újratelepítettem a Windowst. A rendszerindítás után a gép többé nem kérdezi meg, hogy Windowst vagy Linuxot akarok-e futtatni, hanem egyből a Windowst indítja. Helyreállítható-e a Linux?
Kunal S Doddanavar, kunal_s_d@indiatimes.com

A Windows vagy eltávolította, vagy letiltotta a Linux rendszerbetöltőjét, ami a Red Hat 7.1 esetén a LILO. Indítsd a rendszert a helyreállítólemezről, fűzd be a Linux lemezrészét például így: `mount /dev/hda1 /mnt`, és futtasd a `run lilo -R /mnt` parancsot a rendszerindítás előtt. Ha GRUB-ot használtál, a `grub-install` oldja meg a gondot.
Marc Merlin, marc_bts@google.com

Az újabb Red Hat-terjesztéseknél – amelyek a GRUB rendszerbetöltőt használják – indítsd el a rendszert a helyreállítólemezről, és telepítsd újra a GRUB-ot a `grub-install` paranccsal. Ha nem készítettél helyreállítólemezt, akkor a rendszert a telepítő CD-ről indítsd, helyreállító módban.
Christopher Wingert, cwingert@qualcomm.com

Linux Journal 2003. augusztus, 112. szám

Új termékek

SmartFLeX SFT-CXC Network Terminal

Az SFT-CXC karakteres és grafikus terminál a SmartFLeX Technologies beágyazott Flash Linux-rendszerén alapul. Az SFT-CXC karakteres terminálüzemmódja akár öt egyidejű munkamenet is támogat – teljes képernyős üzemmódban, etherneten vagy soros kapcsolaton keresztül. X-terminálként használva egy XDMCP-munkamenetet használhatunk a hálózati gazdagéphez. A különféle ablakkezelők használatára felkészítő bővítményeket mellékelték a programhoz. Az SFT-CXC egy webböngésző program segítségével a távolból is beállítható.

Adatok: SmartFLeX Technology, Inc., 623 Selvaggio Drive, Suite 220, Nazareth, Pennsylvania 18064, telefon: 610-746-2390, ➲ http://www.smartflextech.com

ATG 6

Az ATG 6 a Red Hat Advanced Server 2.1-re telepíthető e-kereskedelmi, üzleti folyamatokat és kapcsolatokat kezelő alkalmazáscsomag. A program önkiszolgáló rendelésfelvételre, számlakezelésre és vevőoldali feladatok (például termék-összehasonlítás, ajándékküldés és fizetés) végrehajtására képes. A projektek logikai munkafolyamatát önműködő eszközök irányítják, a kölcsönhatások teljes sorozata önműködően megy végbe. Az egymáshoz kapcsolódó modulok kezelik a közzététellel, kereséssel, elemzéssel, fizetéssel és a csalások elleni védekezéssel kapcsolatos feladatokat. Az ATG 6 többféle meglévő vállalatirányítási és vevőkapcsolati rendszerrel összekapcsolható.

Adatok: ATG, 25 First Street, Second Floor, Cambridge, Massachusetts 02141, telefon: 617-386-1000, ➲ http://www.atg.com

Trustix Small Office Server

A Trustix megjelentette a Trustix Small Office Servert, amelyet legfeljebb 25 hálózatba kötött felhasználót kiszolgáló környezetbe terveztek (frissíthető az ötven felhasználós változatra). A Small Office Server tartalmazza a Trustix terjesztést, és web-, levél-, proxy- és helyi hálózati kiszolgálóként használható. Telepíthető a meglévő gépekre, vagy előre telepített formában megtalálható az IBM xSeries gépein. A terjesztésnek része a RAV vírus- és levélszemétirtó programja, valamint a NetVault biztonsági mentéseket készítő alkalmazása. A Small Office Server támogatja a felhasználók állományainak központosított tárolását, a hálózati gyorsítótárazást és a központosított beléptetést.

Adatok: Trustix, 4819 Emperor Boulevard, 4th Floor, Durham, North Carolina 27703, telefon: 919-313-4599, ➲ http://www.trustix.com

Interphase IPSec-gyorsítókártyák

Az Interphase új hálózatbiztonsági termékvonalának első képviselői a 45 NS (PMC) és az 55 NS (PCI) hálózatbiztonsági gyorsítókártyák. A kártyák alkalmazásával megszüntethető a VPN-ek, átjárók, útválasztók és tűzfalak szűk keresztmetszete, mert a kártyák a számításigényes IPSec-feldolgozást a számítógép processzora helyett elvégzik. A gyorsítókártyák képesek a fejlécelemzésre, a hasznos forgalom kiemelésére, tömörítésre, titkosításra, hitelesítésre és csomagok összeállítására. Mindkét kártya 500 Mb/s sebességű 3DES-átvitelre képes, és gyorsítja a DES, MD5, SHA-1, RC4 és AES biztonsági algoritmusokat. További jellemzők: 66 MHz-es PCI-sín, 64 MB saját memória, teljesen kétirányú OC3-sebességek támogatása és 512 K egyidejű munkamenet.

Adatok: Interphase, Parkway Centre, Phase 1, 2901 North Dallas Parkway, Suite 200, Plano, Texas 75093, telefon: 800-327-8638, ➲ http://www.interphase.com

SnapGear Embedded Linux

Új, ingyenes beágyazott Linux-terjesztést adott közre a SnapGear Inc. A SnapGear Embedded támogatja az MMU nélküli processzorokat (például ColdFire, ARM és SPARC), valamint az MMU-val rendelkezőket, többek között a SuperH-t, az Xscale-t és a x86-ot is. A terjesztés a SnapGear által karbantartott μClinux-foltokon alapul, segédprogramokat, szabványos API-kat és programkönyvtárakat tartalmaz az egyedülálló programokhoz, valamint forráskódgyűjteményt. A SnapGear Embedded ingyen letölthető a webhelyéről, de megfelelő díj ellenében CD-lemezen is elérhető.

Adatok: SnapGear, Inc., 7984 South Welby Park Drive #101, West Jordan, Utah 84088, telefon: 801-282-8492, ➲ http://www.snapgear.com (company site), ➲ http://www.snapgear.org (downloads).

3DBOXX M4 Opteron munkaállomások

A BOXX Technologies bejelentette térbeli leképező munkaállomásainak új családját, amelyek két Opteron processzort tartalmaznak. A 240-es, 242-es és 244-es processzorokkal felszerelt gépek már kaphatóak. Az M4-es munkaállomás nVidia Quadro rendszert használ a térbeli tartalom leképezésére és mozgatására a Maya, 3DS Max, Softimage XSI, LightWave 3D és Houdini programokkal. Az alapkiépítésű munkaállomás tartalmaz egy AMD-8111 HyperTransport PCI-alagutat, egy AMD-8151 HyperTransport AGP-alagutat, 128 bites kétcsatornás memóriasínt, legfeljebb 8 GB ECC-hibajavítással ellátott 333 MHz-es DDR-memóriát, négy DIMM-bővítőhelyet, kétcsatornás UltraDMA 133 IDE-vezérlőt és hatcsatornás hangot. A munkaállomások egyéni felszereltséggel is megrendelhető. A jó hőelvezetés érdekében a gépek könnyű alumíniumházban vannak, és két 92 mm-es ventilátor fújja bennük a levegőt.

Adatok: BOXX Technologies, Inc., 10435 South Burnet Road, Suite 120, Austin, Texas 78758, telefon: 877-877-2699, ➲ http://www.boxxtech.com

Linux Journal 2003, 112. szám

Hogyan teszi a Linux okosabbá a vállalatokat? (2. rész)

Hogyan is változtatja meg a Linux a hagyományos vállalatok információtechnológiáját?

Nézzünk meg például egy levelet, amelynek a szerzője névtelenséget kért: „Egy Fortune 50-be tartozó vállalatnál dolgozom, ahol az IT-részleg új nyílt forrású irányvonalat léptetett érvénybe. Emellett egy jóváhagyott nyílt forrású eszköztárat is kiadott (saját, belső „márkanév" alatt), amit az alkalmazottak egy weblapról tölthetnek le. Már a 3.0 változatnál tartunk, ami mind Unix, mind Windows alatt megtalálható. Az eszköztár Unix-oldalon GNU és egyéb eszközökből tevődik össze, Windows-oldalon pedig a Cygwin eszköztárat tartalmazza.
A Linux (jelenleg a Red Hat) az egyike az „eszközöknek".
Az eszköztár-projektet vezető srác szintén a szabad programok szószólója.
Az irányelv kinyilatkoztatott célja, hogy lehetőséget adjon a vállalatnak a megtakarításokra. Nyilvánvaló cél az is, hogy lehetőséget biztosítsanak a vállalatnak szellemi tulajdonuk megvédésére egy harmadik fél (ez a Microsoft) manipulációjával szemben.
Az irányelvnek része az úgynevezett „ne kérdezz, ne mondd el" elv. Az IT beleegyezett, hogy nem panaszkodik, ha valaki elhatározza, hogy ezeket az eszközöket bármelyik, a felhasználó által használt számítógépre telepíti.
Az összes nem vezetői beosztásban dolgozó alkalmazottnak most engedélyt kell kérnie helyi vezetőjétől az eszköz letöltésére és telepítésére. A vezetőknek nincs szükségük ilyen engedélyre, hogy az eszközt a saját gépükre telepíthessék.
Az irányelv szerint az IT semmilyen módon sem szól bele a dologba. Így a linuxos eszközök és a többi szabad program terjedése természetesen és egységesen fog megtörténni.
Mielőtt ezt az irányelvet létrehozták volna, még a GNU Emacs programnak is át kellett esnie a hivatalos IT-felülvizsgálati, illetve -engedélyezési eljáráson. Még akkor is, ha a főnököd akarta, az IT harcolhatott ellene, ha kedve volt. Az új irányelvvel azonban mindez már csak történelem."
A szervezetben az irányelv elkerülhetetlenül alkalmazkodik a fejlesztési életciklushoz. Ma ezek a trendek a Linuxnak kedveznek – nagy időket élünk. A Linux-fejlesztőkről szóló legújabb felmérés szerint (mely fejlesztők más felületeken is dolgoznak) az Evans Data Corp. gyors átrendeződést figyelt meg a kereskedelmi Unixtól és Windowstól a Linux irányába: „Az esetek negyven százalékában a Linux az elsődleges választás. A Windows 2000-nek nagyon erős, 29 százalékos részesedése van, amit a Windows XP követ 12 százalékkal. A helyzetkép azonban folyamatosan változik. A következő évben a válaszadók száma, akik a Linuxot elsődleges fejlesztési felületként használják, 15 százalékkal nőni fog, vagyis negyvenről 55 százalékra."

Forgalmazók: azt nyújtani a vásárlóknak, amit ők szeretnének

A forgalmazók új kényszerítő ereje az, hogy azt adják a vevőiknek, amit ők szeretnének, és ne próbálják meg belekényszeríteni őket egy választás nélküli kapcsolatba. Köszönik, ebből már elegük van. ***Mårten Mickos***, a MySQL (➲ http://mysql.com) ügyvezetője hisz abban, hogy az olyan vállalatok, mint az övé is, képesek felébreszteni az IT-tudatosságot. Azt nyilatkozta, hogy a MySQL azért tudja sikeresen árulni a szabad programot (a MySQL GPL felhasználási szerződésű), mert „több az érték a látható kódok fejlesztésében, mint abban a forráskódban, ami nem hozzáférhető; egy óriási vevő- és fejlesztői közösség tagjai vagyunk, mind szenvedélyesen fejlesztjük az alapkódot. Minden nap bebizonyítjuk, hogy létezhet olyan kereskedelmi kapcsolat, amelyik szabad programot eredményez."
Ez a kereskedelmi kapcsolat még mindig nem az, ami a legjobb forgalmazóknál és a vevők bürokráciájában megtalálható. A nagy vevőknél mindez mindkét oldalon középszinten megy végbe.
A MySQL vevői listája lenyűgöző, köztük van a Nokia, a Yahoo, a NASA, a Silicon Graphics és a Cisco. ***Jeremy Zawody***, aki magát „Yahoo-technikusnak" nevezi, a Yahoo gazdasági osztályán dolgozik. A következőket mondja: „A MySQL ahhoz hasonlóan fog beszivárogni a vállalathoz, ahogyan a Microsoft SQL Server tette, de annál sokkal nagyobb sebességgel. A MySQL olyan az Oracle-nek, mint a Linux a Windowsnak. Lassan, de határozottan felkúszik a táplálékláncon, ugyanúgy, mint a Linux tette." Amikor megkérdeztem Mårten Mickost, hogy vajon a MySQL versenyzik-e az Oracle-lel, azt válaszolta, hogy nem. „Inkább helyettesítjük az Oracle-t, mintsem versenyzünk vele."
Wim Coekaerts (➲ http://otn.oracle.com/oramag/Coekaerts.html), az Oracle linuxos magfejlesztő csoportjának vezetője a következőket nyilatkozta: „A Linux nagyon, de nagyon fontos az Oracle számára. Szinte már linuxos cég vagyunk." Büszkén állítja, hogy magfejlesztői csoportja hozzájárult ahhoz, hogy Linuxot „vállalati szintűvé" tegye.

Történet az új szabványról

Az Oracle-hez hasonlóan a többi forgalmazónak is alkalmazkodnia kell ahhoz a világhoz, amelyben a Linux és a nyílt forrás követői az alapvető IT-hátteret alkotják. Ez a háttér gyorsan válik szabvánnyá, akárcsak a „százas" szeg és a csavarhúzó. A forgalmazók mindig üdvözölni fogják e háttér előnyeit, és hozzájárulnak a fejlődéséhez, de a kapcsolatuk formája az elvonttól el fog tolódni a kézzelfogható felé, az elvont játéktérről a valódi IT-projektek felé, ahol valami hasznoshoz járulhatnak hozzá. Amikor ez meg fog történni és a programipar felnő, az érdem végre oda fog kerülni, ahova már régóta kellett volna: az okos emberekhez, akik elkezdték használni a Linuxot, hogy vállalataikat okossá tegyék – függetlenül attól, hogy ezek a vállalatok mivel kereskednek.

Linux Journal 2003. július, 111. szám

Doc Searls (doc@ssc.com)
A Linux Journal szerkesztője és a Cluetrain Manifesto társszerzője.

A kisegítő lehetőségek szempontjából nézve

Hogyan állja meg a helyét a Linux a fogyatékkal élő felhasználók támogatása terén?

Az angol accessibility kifejezés a fogyatékkal élőket és a csökkent képességűeket támogató, az általuk is használható megoldásokra vonatkozik, ezeket magyarul talán a „kisegítő lehetőségek”-nek nevezhetnénk. A kérdés most az, hogy mennyire használható a Linux a fogyatékkal élők és a csökkent képességűek számára?

A válasz ma még nagymértékben attól függ, hogy milyen fogyatékról van szó. A kisegítő lehetőségekkel kapcsolatos kihívások megoldását célzó egyre egységesebb törekvések mögött talán maga a linuxos szellemiség a legfontosabb hajtóerő. A nyílt forrású és szabad programok világának alapját olyan közösségi értékek alkotják, amelyek senki kizárását nem engedik meg. Mindenkinek joga van használni a programot, mindenki megnézheti és módosíthatja a kódot – miért maradna ki valaki csak azért, mert nem tud képernyőről szöveget olvasni, vagy nem tudja egy időben lenyomni az ALT és az F1 gombot?

A Linux tényleg példamutató eredményeket mutat föl a kisegítő lehetőségek terén. Többnyire azonban a kisegítő lehetőségek támogatása sajnos csak esetleges – ez egyrészt annak a ténynek a mellékterméke, hogy a Linux gyökerei mélyen az ASCII-kód világába ágyazódnak, másrészt annak, hogy a Linux kimagaslóan jól képes szinte bármilyen eszközről bemenetet olvasni. Nem mintha a fejlesztők szándékosan hagynák ki a kisegítő lehetőségek támogatását – egyszerűen csak ezt a szempontot mindeddig nem alkalmazták a kódellenőrzés során.

Egy másik hatóerő a szóban forgó programoknak elsőbbséget biztosító törvények és irányelvek megjelenése. Ezek közül a legjelentősebb az amerikai kormány beszerzési gyakorlatát szabályozó 508. bekezdés néven ismert jogszabály. E nemrég szigorított rendelkezés szerint az amerikai kormány köteles kisegítő „elektronikus és informatikai megoldásokat” alkalmazni a munkahelyeken és a nyilvánosan elérhető elektronikus tájékoztatási rendszerekben, amennyiben ilyen technológia rendelkezésre áll. Az 508. cikkely hatására megnőtt az érdeklődés a kisegítő megoldások iránt, egyszerűen azért, mert az amerikai kormány az egyik legjelentősebb vevő az informatikai piacon – jelenleg mintegy negyvenmilliárd dollárt költ évente erre a célra, és ez a szám a várakozások szerint mindössze öt év alatt ötven százalékkal nőhet. Hozzátéve ehhez azt a megbecsülést, amit a linuxos szakemberek a közösségi értékek iránt mutatnak, szorongató érzés arra gondolni, hogyha csak hajszál híján is, de a Linux esetleg mégsem felel meg a társadalmi elvárások e mércéjének.

Az utóbbi évek során számos fejlesztő dolgozott – elsősorban a Sun Microsystems, de az IBM, a Red Hat és a Ximian támogatásával is felvértezve – az új Gnome 2.0-s környezet kisegítő keretszolgáltatásainak módszeres kidolgozásán. Őszintén szólva nehéz elképzelni, hogy ez az 508. cikkely nélkül is megtörtént volna-e, de ezeknek az erőfeszítéseknek a haszna messze túlterjed az Egyesült Államok határain. A Gnome kisegítő szolgáltatásainak honlapja (➲ http://developer.gnome.org/projects/gap) jelenleg talán a legjobb forrás arról, hogy mit is jelent és hogyan valósíthatók meg a kiegészítő lehetőségek a Linux esetében.

Az egész kérdéskör ugyanakkor be- és kiviteli kérdésként is összegezhető.

A gondok – bármilyen sokfélék és összetettek számtalan megjelenési formájukban – alapvetően egyszerűek.

Bármilyen beviteli módról legyen is szó, amin keresztül egy számítógép adatokat fogadhat a felhasználóktól, akadnak olyanok, akik képtelenek használni őket. Ugyanígy minden emberi fogyasztásra szánt kiviteli módhoz találhatók olyan felhasználók, akik az adott közvetítő eszközön keresztül nem képesek befogadni a kimenetet. A kérdés, hogy miként lehet ezt az alapvetően bináris elven működő gépet rávenni arra, hogy igény szerint bármilyen eszközről bemenetet fogadjon, és miként alakítható a kimenet megjelenési módja úgy, hogy a lehető legszélesebb felhasználói kör számára értelmezhető legyen.

Az erkölcsi kötelesség nyilvánvaló. A közösségi programok csak akkor lehetnek valóban közösek, ha mindenki számára elérhetők. Lehetséges, hogy ez az igény bizonyos területeken akadályt jelent majd. Többnyire azonban nagyon kevéssé zavaró, egész egyszerűen azért, mert a Linux foglalatokat (sockets) használ. A munka javát nyilvánvalóan erre szakosodott fejlesztők fogják elvégezni, mivel a jól működő megoldások elkészítéséhez sok különleges ismeret szükséges. De amit meg akarnak valósítani, az nem különbözik attól, amit mindannyian akarunk – és vissza is érkeztünk a közösségi szellemhez.

Linux Journal 2003. augusztus, 112. szám

Janina Sajka
Az Amerikai Alapítvány A Vakokért (American Foundation for the Blind, AFB) műszaki kutatási és fejlesztési igazgatója.

A hihetetlen Hulk erejétől az ILM Halálcsillagáig

A július elejétől a magyar mozikban is bemutatott Hulk című film szintén az ILM linuxos leképezőtelepén készült.

A Linux Journalban (és a Linuxvilág 2002. augusztusi számának 32. oldalán) megjelent történet – amelynek témája az Industrial Light & Magic (ILM) Linuxra való átállása volt asztali gépeiken és leképezőtelepükön – rekordmennyiségű olvasói visszajelzést váltott ki.
A Csillagok háborújának Halálcsillaga mégis létezik! Bár nem mint fenyegető, bolygókat megsemmisítő fegyver, hanem mint az a leképezőtelep, ami az Industrial Light & Magic műhelyében készülő számos mozifilm hátterében húzódó számítási teljesítményt adja.
„A Linux folytatja töretlen fejlődését" – állítja ***Cliff Plumer***, az ILM műszaki igazgatója. „Leképezőtelepünk jelenleg több mint 1500 processzorral bír, és az asztali gépek révén minden este csaknem 1000 továbbival gyarapszik." A leképezőtelep mind a kizárólag erre a célra használt processzorokat, mind pedig az üresjáratban dolgozó asztali gépek teljesítményét. kiaknázza

1. kép Az ILM RackSaver linuxos leképezőtelepe jelenleg 1500 processzorral bír, és a Star Wars Episode III elkészítéséhez ez a szám a kétszeresére fog emelkedni

A Halálcsillag leképezőtelep

„Leképezőtelepünk szinte teljes egészében linuxos gépekből áll, amióta csak SGI gépeinket lecseréltük" – tájékoztat ***Mike Thompson*** rendszertervező. „A telep körülbelül 750 csomópontból, azaz 1500 processzorból áll." A leképezőtelep 1U kétprocesszoros, keretszerelésű sorba kötött számítógépekből áll. Hagyományos értelemben véve ez nem egy szuperszámítógép – mindegyik félig függetlenül, hálórendszerbe kötve működik, nincs meg az egyetlen munkafolyamat futtatásának kötöttsége, mint egy szuperszámítógép esetén. Itt az ILM-nél egy saját, parancsfájl-ütemező program, az ObaQ vezérli az egyes gépekre jutó munkát.
Ahogy a rendszerek nyers teljesítménye növekszik, egyre nagyobb az ILM géptermének elektromosság- és hűtésigénye. „Fontos, hogy csökkentsük a fogyasztást és a hőmérsékletet – világít rá Thompson. Mi AMD Athlon 1600-as processzorok mellett döntöttünk, olyan alacsony fogyasztású változat mellett, amit nem könnyű beszerezni." Mindegyik csomópont 2 GB memóriával rendelkezik, ami 4 GB-ra bővíthető, és a csomópontok a két processzor kihasználása végett egyszerre két munkafolyamatot futtatnak, de ez néha egy folyamatra csökken, amikor a teljes memóriát ki szeretnénk aknázni.
Az AMD-vel felszerelt csomópontok a RackSaver cég RS-1100-s egységei. A RackSaver a National Association of Broadcasters (NAB, azaz Műsorszóró Társaságok Nemzeti Egyesülete) kongresszusán keltette fel az ILM figyelmét, és kapta meg azt a lehetőséget, hogy ajánlatot tegyen a leképezőtelep megépítésére. A RackSaver legnagyobb ellenfele a nehézsúlyú IBM és a Dell. A cég előnye ***David Diggers*** vezérigazgató szerint abban áll, hogy a versenytársakéhoz képest kétszeres sűrűségű kiszolgálókat képesek gyártani. „A piacnak ebben a szeletében az ILM, Pixar és Warner Brothers stúdiókkal kötött üzleteink révén nagyon erősek vagyunk" – büszkélkedik David.
A RackSaver telep kiszolgálói 100BASE-TX csatolókon keresztül csatlakoznak egy Foundry 8000 hálózati kapcsolóhoz, amely ezeket a kapcsolatokat egy gigabites központi hálózati kötelékké összesíti. „Nemrégiben bővítettük ki a hálózati központot egy 10 gigabites összeköttetéssel, ami nagyon sokat segített – meséli Thompson. Egy fájlkiszolgáló-maggal és 2500 leképezőközponttal rendelkezünk, ez napi mintegy 70 TB összesített forgalmat jelent."

Mitől rettenetes a zöld szín?

„A Hulk nem egy szokványos képregény-mozi" – világosít fel ***Doug Sutton*** műszaki igazgató. „Évek óta ez jelentette az egyik legnagyobb kihívást a munkáink közül. Egy 15 láb magas fickót élethűen megalkotni – ez igazi erőpróbát jelent." A Hulk egy számítógéppel előállított digitális színész, bonyolult érzelmekkel és zöld színű bőrrel.
„A filmeket úgy gyártják, hogy az emberek ne nézzenek ki zöldnek, vagyis a zöld színt eltolják – magyarázza ***Rod Bogart***, a vezető programmérnök. „Mi Kodak Premiere nyomtatókészletet használunk, ennek mélyebbek a színei, mint a Kodak Visionnek. Ha a szereplő zöld, ahogy Hulk is, elég nehéz olyan látványt létrehozni, ahol a bőr színe tényleg zöldnek hat.
A Jurassic Park zöld dinoszauruszai könnyebb feladatot jelentenek. A dinoszauruszoknak nincsenek olyasfajta fényfoltjaik a bőrükön, mint az embereknek." A nézők kevésbé megbocsátóak az emberi arccal kapcsolatban – főleg ha az még zöld is.
„A zöld a fehérhez közelítve sárgás árnyalatot vesz fel – teszi hozzá Sutton. Muszáj egy megjelenítőn is ellenőrizni, mielőtt az ember elfogadná vagy elvetné".
Az élő jeleneteket San Francisco utcáin rögzítették. ***Jennifer Conelly*** egy utca közepén áll, és egy nagy zöld fickónak játszik, aki nincs is ott. Egyetlen segédeszköze egy pózna, a tetején egy zöld fejjel. „Készítettek néhány igazán jó valós hatást, robbanást, de főleg számítógépes grafikáról van szó – magyarázza Sutton –, a Maya és a Softimage keverékéről".

A Cineon és az OpenEXR

Egy film képeinek a rögzítése nagyobb dinamikatartományt igényel, mint amit a JPEG vagy a PNG támogat. A Kodak Cineon hosszú ideje a digitalizált film szabványos formátuma. A Cineon 10 bites logaritmikus formátum szemben a JPEG-gel, ami 8 bites lineáris (azaz 24 bites RGB), így a Cineon nagyobb dinamikával és – különösen a fekete közelében – gazdagabb színkészlettel rendelkezik. Ahogy növekedett a számítási teljesítmény, és a legtöbb számítás a lebegőpontos ábrázolásra

2. kép Az ILM saját ObaQ ütemezője éppen egy Linux-parancsfájllal vezérelt leképezést futtat a Hulk című film készítése során

3. kép Michael Thompson az ILM-től a 20 terabájtos EMC fájlkiszolgálóval

váltott át, a 10 bites logaritmikus formátumban való munka vált korlátozó tényezővé. Az OpenEXR egy új, lebegőpontos képformátum.

„Az OpenEXR formátum egy film jobb digitális ábrázolását biztosítja, mivel több mint 30 f-egység dinamikatartománnyal rendelkezik, anélkül, hogy veszítene a pontosságából – fejti ki a véleményét Plumer. A korábbi 8 bites fájlformátumok csak körülbelül 7–10 f-egységgel rendelkeztek, és különösen nagy kontrasztú képeket már nem voltak képesek megfelelő módon megjeleníteni."

Az ILM 2000 nyarán alkotta meg az EXR formátumot, és olyan filmek elkészítése során használta, mint a ***Harry Potter és a bölcsek köve***, a ***Men in Black II***, a ***New York bandái***, a ***Jelek*** (Signs), az ***Álomcsapda***, a ***Hulk***, a ***Van Helsing***, a ***Peter Pan***, az ***Idővonal*** és a ***Karib-tenger kalózai***.

Az Open EXR veszteségmentes tömörítést használ, mint a PNG, nem pedig a JPEG-hez hasonló veszteséges tömörítést. Valójában létezik egy kihasználatlan Piz12 veszteséges tömörítési lehetőség is. A PNG-vel és JPEG-gel ellentétben az OpenEXR hullámkódolást használ, ami lényegében egy olyan faszerkezet, ami a képpontok közti előjeles különbségeket tárolja. Mivel a számok nagyságrendje kisebb és kevesebb egyedi érték fordul elő, a Huffman-kódolás nagyobb hatékonysággal tudja összetömöríteni az adatokat. Az EXR támogatja a 32 bites lebegőpontos, a 32 bites egész és 16 bites lebegőpontos ábrázolást, tetszőleges számú csatorna esetén.

Az egyes csatornák különböző színmélységgel rendelkezhetnek, például az RGBAZ képek esetén nagyobb Z-mélységre van szükség – a jellemző érték a 16-16-16-16-32. A Z-csatorna fizikai mélységet jelöl, nem pedig egy szín- vagy alfamaszkot, képzeljük el inkább úgy, mint valami szonárt. 2003 januárjában az ILM kibocsátotta a nyílt forrású OpenEXR-t. Az első olyan nyílt forrású program, ami támogatta ezt a formátumot, a CinePaint volt.

A CinePaint

A CinePaint, amit a közelmúltig FilmGimp néven emlegettek, egy képenként haladó mozgófilm-retusáló rendszer, ami 1998-ban vált ki a Gimpből. Számomra is váratlan módon lettem a CinePaint-projekt vezetője, miután a Linux Journal számára írtam néhány cikket a FilmGimpről. A megszokott állókép-formátumokon túl a CinePaint támogatja a filmgyártásban elterjedten használt fájlformátumokat is. Ezek között található a Cineon, az RnH 16 bites lebegőpontos formátum (a Rhythm & Hues által létrehozott formátum, ami a 32 bit felének a levágásával jött létre), a Radiance HDR, a LogLuv Tiff és most már az Open EXR. A CinePaint OpenEXR bővítménye az ILM munkája.

Az SDev, a LUT és a rácsvonalak

„Soha nem akarunk egy képet a megfelelő SDev nélkül megnézni" – jelenti ki Sutton. Nagyon fontos, hogy azt lássuk, amit a folyamat végén eredményként kapunk. Az SDev egy szimulációs eszköz, amivel a monitort a filmhez hasonlatossá tudjuk tenni. Az OpenEXR-re való áttérés óta nem használunk LUT-okat többé, de SDev-eket igen." A LUT egy keresési tábla, amelyet egy kép gammájának a kijavításához használtunk korábban. A monitorok fényereje nem arányos a bemeneti feszültséggel, inkább a bemeneti feszültség valamelyik hatványával. Ezt a kitevőt nevezzük gammának, amelynek az értéke a kijelzőtől függ. A Macintosh gépek jellemző értéke 1,8, ami a PC-k esetén 2,2 körüli.

„A LUT-ok működésének elve leginkább a kontraszt változtatására épül – magyarázza Bogart. A LUT ellenben nem képes a színtelítettség növelésére vagy csökkentésére." A LUT helyett az ILM-nél egy összetettebb rácsvonalas számítási módot alkalmaznak. A rácsot 13×13×13 méretű térbeli kockáknak kell elképzelnünk – tulajdonképpen háromdimenziós indexelt tömbről van szó. Egy páratlan számot használnak, a középpont szürke. A rácsvonalas ábrázolásban minden RGB érték leképezésre kerül, szemben a LUT-tal, ami csatornánkénti leképezést végez. Három egymástól független keresési tábla alkalmazása nem megfelelő egy film látványának visszaadásához – különösen telített zöld szín esetén nem. A rács egy 64 K bejegyzést tartalmazó számított tábla segítségével állítja be a 12 pontos filmgörbének megfelelő értéket. Egy 0 és 1 közé eső rácsindex háromvonalas vagy négyoldalú interpoláció alkalmazásával és az ezt követő gammakorrekciójával szolgáltatja a három értéket. A folyamat lassabb, mert minden egyes képpontot a többivel együtt kell kezelni, de élethűbb, mint az RGB-

A lebegőpontos számábrázolási formátumok összehasonlítása

Formátum	Összes bitek száma	Előjel	Kitevő	Mantissza
IEEE 754 Double	64	1	11	52
IEEE 754 Float	32	1	8	23
nVidia/ILM Half	16	1	5	10
RnH Float16	16	0	8	8

csatornákon alapuló kikeresés. „A rácsos módszer nem csak a Hulk esetén alkalmazható eredményesen” – mutat rá Bogart. A ***Minority Report*** esetén egy színtelenítő folyamat látványát szimuláltuk a rácsmódszerrel. A LUT színtelítetlenség előidézésére sem alkalmas.”

GPU-programozás az nVidia Cg-vel

A nyers 16 bites OpenEXR adatformátumot Halfnak (fél) is nevezik – utalva a 32 bites lebegőpontos ábrázolás megfelezésére. A Half adatformátum az nVidia grafikus kártyák belső adatformátuma. Jó lenne, ha a rácsmódszer számításai, amik jelenleg a központi feldolgozó egységet terhelik, közvetlenül a videokártya processzorán (GPU) futhatnának. Ennek megvalósítása a grafikus kártyák fejlődésével egyre közelebb kerül. „Már nagyon várjuk, hogy ez lehetővé váljon” – vallja be Bogart. A képpel kapcsolatos számításokat a GPU képpontárnyaló rendszerére szeretnénk átterhelni.” Az nVidia egy új, C-szerű fordítót, illetve programkönyvtárat kínál, a Cg-t, képpontkódok – közkedvelt nevükön „árnyalók” (shaders) – futtatására a GPU-n. Az ATI hasonló technológiát kínál High Level Shading Language (magas szintű árnyaló nyelv) néven, a 3Dlabs pedig az OpenGL Shading Language fejlesztésén dolgozik.
A GPU programozása hasonló a beágyazott rendszerek programozásához, ahol a kódot a gazdafelületen fordítjuk, majd ezt töltjük le a beágyazott rendszerre. A GPU-programok futásidőben fordíthatók és tölthetők át a grafikus kártyára. A fordítóprogram (compiler) a futásidejű programkönyvtár részét képezi. Néhány 3D-programcsomag, például a Softimage és a Maya már kezdi használni a Cg-t a leképezési teljesítmény növelésére.

Modellezés és leképezés

Az Alias/Wavefront Mayát korábban a részek (particles) és néhány szereplőmozgási modell kidolgozására használták. A Pixar RenderMan és a Mental Images cég Mental Ray programja végezte a leképezést. A sugárkövető leképezők tükröződő felületei jobbak, de az előállításuk hosszabb időt vesz igénybe. Hála a gyorsabb és olcsóbb Linux-rendszereknek, a módszer gyakorlati alkalmazása egyre egyszerűbbé válik. Mind a RenderMan, mind pedig a Mental Ray támogatja az árnyaló programozást a képek egyedibbé tételére. A RenderMan könnyen elsajátítható saját árnyaló nyelvvel rendelkezik. A Mental Ray a C nyelvet használja, ami több munkát igényel, de hatékonyabb. Hogy melyik programot használjuk, azt jelenetenként eldönthetjük, minden jelenet leképezése egy ütemező parancsfájl vezérlése szerint történik.

Az ObaQ ütemező parancsfájl

*„Florian Kine*s – aki az OpenEXR megalkotásának hátterében is áll – írta az ütemező parancsfájlunkat, sok más mellett, amit az SGI Irix óta alkotott – tudatja velünk Hess. Jó hasznát vettük a nagygépnek és az asztali számítógépeknek. Amikor elkezdtük a Linuxra való átállást, jobb erőforrás-gazdálkodást szerettünk

4. kép Az ILM-nél készült Hulk weboldala

5. kép Az ILM-nél készült Terminátor 3 weboldala

volna megvalósítani.” Az ObaQ első változata jelenetek szerint különválasztotta a gépeket – ez bizony nem túl hatékony erőforrás-kezelés.
„Az ObaQ kiváltására indított IMP projektünk, ami egy központosított erőforrás-kezelő rendszert állított volna az ObaQ helyére, nem járt sikerrel – mondja Hess. Visszatértünk az ObaQ-hoz és ennek Linuxra való átültetéséhez, ami körülbelül két hétig tartott. Három vagy négy hónappal ezelőtt Florian úgy döntött, megoldja, hogy bármelyik jelenet bármelyik gépet használhassa.” Az ObaQ egyenrangú (peer to peer, P2P) ütemező rendszer. Ennek az az előnye, hogy egy esetleges kiszolgálóhiba esetén az egész rendszer nem kerül kapcsolat nélküli állapotba. Az ObaQ2 a teljes ütemezés megvalósítására egyetlen gépet használ – a futtatására tanácsos egy független gépet használatba venni. Az ObaQ2 kiszolgáló elvesztése nem jelenti az egész géppark leállását. Léteznek más ütemező rendszerek is, mint a népszerű zárt forráskódú termék, a Platform LSF, vagy a nyílt forrású Condor és OpenPBS ütemezők, de az ILM az ObaQ használatának a folytatását tervezi.
Az SGI a folyamatok megfigyelhetősége céljából hozzáadott néhány képességet az Irix-rendszermagjához, ezáltal a processzoridő és az ideiglenes helyek követhetővé váltak. Ezektől az értékektől függ, hogy az ILM gépidejét hogyan

adja vissza a központi elszámolás a projektek számára. Az ILM azt tapasztalta, hogy a Linux */proc* fájlrendszer nem biztosítja maradéktalanul ezeket a statisztikákat, vagy az rendkívül sok többletmunkával jár, és az ObaQ-t sem támogatja változtatások nélkül.

A Halálcsillag rendszermagjának módosítása

„Florian arra kért, hogy foglalkozzam a Linux-rendszermag néhány kérdésével – meséli Hess. Az egyik dolog, hogy a Linux nem ad módot a futási szálak (thread) és a folyamatok (process) megkülönböztetésére. A `ps` minden szálat külön folyamatként tüntet fel." Néhány munkafolyamat, amilyen a Mental Ray is, keretenként több szálat is képes futtatni egymással párhuzamosan. A Linux `top` vagy `ps` parancsa minden szálat úgy mutat, mintha 1 GB RAM-ot foglalna le, pedig az csak olyan osztott memória, ami többszörösen lett figyelembe véve. A Linux arról sem képes adatokat szolgáltatni, hogy melyik munkafolyamat nyit ideiglenes fájlokat. Az ObaQ-nak szüksége van ezekre az ismeretekre ahhoz, hogy egy munkafolyamat félbeszakításakor ezeket az ideiglenes fájlokat is törölhesse. Hess egy linuxos magmodult hozott létre, hogy csapdát állítson a nyitott állományoknak, elágazásoknak, másolatoknak, kijáratoknak és átnevezéseknek, abból a célból, hogy lehetővé váljon a pontos kimutatás elkészítése. A rendszermagmodul a munka nagy részét elvégzi, de a hurkolt (hooked) hívásoknak minden olyan munkafolyamatot figyelmen kívül kell hagyniuk, amit nem az ObaQ futtat. Ehhez a rendszermag módosítására van szükség. „A `ptrace` zászló (flag) egyik használaton kívüli bitjét használtam fel – magyarázza Hess. Minden x86 munkafolyamat rendelkezik egy 32 bites `ptrace`-vektorral. A 2.4.20-astól kezdve 10 bit jelzi a `ptrace` módot, mint amilyen a lépésenkénti végrehajtás. Valamikor a múlt év folyamán a Linux vagy a glibc megváltoztatta a `ptrace` zászló működését, így az elágazásoknál törlődik. Azt tapasztaltam, hogy a rendszermag törli ezeket a biteket, és megtartja a 32. bitet." Hess elmondása szerint a 2.5 rendszermagban az `OPROFILE` tulajdonság továbbfejlesztett elszámolási (accounting) képességekkel rendelkezik, így esetleg nincs szükség a módosítására. A `ptace` zászló (flag) használaton kívüli bitjének felhasználása gyors módja annak, hogy a munkafolyamatokat az ObaQ-feladatokhoz tartozónak tüntessük fel. „Ez az egyik nagyszerű dolog a Linuxban – lelkendezik Hess. Mivel rendelkezünk a forráskóddal, saját magunk is kieszközölhetjük ezt a módosítást, ez pedig nagyon gyors megoldás. Nem kell a fejlesztőt egy ilyen egyedi műszaki megoldás fejlesztésébe bevonni, mint azt az Irix esetében meg kellett tennünk."

Gondok az NFS-sel

„Ekkora tűzerővel, amivel a leképezőtelepünk rendelkezik, bármelyik fájlkiszolgálót hanyatt lehetne dönteni" – vélekedik Thompson. A nukleáris robbanások leképezése a Hulkban elég döcögős – ez a fő oka későbbi bánatunknak. Egy grafikusnak könnyű felprocesszoroznia a leképezést (vagyis több processzort hozzárendelnie a feladathoz) egészen addig a pontig, amikor is a fájlkiszolgáló végleg megadja magát. Hétszázszor annyi adatot kell elosztanunk, mint korábban."
Az ILM Sun T3-as lemezegységtömböket használ az NFS kiszolgálására. A Linux NFS-ügyfélként való üzembe helyezése számos kérdést vetett fel, amikor másfél évvel ezelőtt működni kezdett. A Linux NFS UDP-csomaghibája miatt (a 2.4.18-as változattól javítva) a Sun Solaris kiszolgáló néhány órás működés után száz százalékra pörgött fel, és használhatatlanná vált. A Sun saját Solaris rendszermagmoduljával és IP-veremfoltjával sietett a segítségünkre a Linux-hiba behatárolásában. A Linux NFS UDP választásának másik gyötrő kérdése a forgalomvezérlés hiánya. „Amikor túlterhelési gondok kezdenek jelentkezni a fájlkiszolgálónkon, a leképezőtelep szolgálatmegtagadási támadást indít a fájlkiszolgálón – kesereg Thompson. Ismét megpróbáljuk a TCP NFS-t egy Linux-ügyfélgépen, most, másfél évvel később. Jövő héten indul a kipróbálása."
A TCP körülbelül öt százalékos többletterhelést jelent.
Az ILM mostani 20 TB-os tárolókapacitását a jövő évben a kétszeresére emeli. „A Csillagok háborúja III. epizódjára leképezőtelepünket duplájára növeljük – dicsekszik Thompson. Ezt a RackSavertől rendelt 3000 új csomóponttal könnyen megtehetjük, de tönkretehetné a fájlkiszolgálónkat." Thompson azt tervezi, hogy az NFS-kiszolgáló összeomlását egy fürtözött fájlkiszolgálóval fogja megakadályozni – egy Sistina GFS-sel vagy valami hasonlóval; a fájlkiszolgálás így nem korlátozódik kizárólag az ILM berendezéseire.

Napi digitális munkaanyagok (dailies)

Az ILM a világ minden tájára küldi napi munkaanyagait az ILM Conduiton (ILM-csatorna) keresztül, ami egy titkosított SSL-protokollt használó, szabadalmazott fájlküldő rendszer. A Blowfish segítségével mindent kétszeresen rejtjeleznek. Az ILM lejátszói futtathatók Windowson, Macintoshon és létezik egy webalapú Java kisalkalmazás-változat is, ami mindenütt futtatható. „Az MJPEG-A QuickTime a fő mozgófilmtároló formátumunk – mondja Thompson –, de a Conduit bárminek az átvitelére alkalmas, így illesztési adatok (matchmove data), digitális képek, napi munkaanyagok is továbbíthatók. A felhasználók a napi munkaanyagokat szabályos hálózati kapcsolataikon keresztül, linuxos gépeiken játszhatják vissza. Ez valóban lenyűgöző, ehhez korábban SGI berendezésre volt szükség." A napi munkaanyagok számára az ILM 20 TB tárterülettel rendelkezik EMC Clarrion FC4700-as tömbökben, amelyek előterében négyprocesszoros Sun 420R kiszolgálók állnak 4 GB memóriával és gigabites hálózati csatlakozással.

Zöld jelzés a Linuxtól

„A Linux használata számunkra annyit jelent, hogy bármilyen feladat megoldásához hihetetlen mennyiségű számítási teljesítmény áll a rendelkezésünkre – jelenti ki Sutton. A Hulkban a haj és bőr megjelenítésére már nem is tudom, hány mintázatréteget használtunk. Félelmetesen összetett jelenetek megalkotására vagyunk képesek a Linux használatával."
A stúdiók tevékenysége a rendelkezésre álló pénz és idő függvénye – a gyorsabb és olcsóbb Linux több film elkészítését teszi lehetővé.

Köszönetnyilvánítás

Köszönet ***Jimmy Perry***-nek (jimmy@racksaver.com), a RackSaver, Inc. marketingszervezőjének az elgondolásáért és az írás kidolgozásában nyújtott segítségéért.

Linux Journal 2003. augusztus, 112. szám

Robin Rowe (robin.rowe@ movieeditor.com)
A MovieEditor.com internetes és televíziós videoalkalmazásokat készítő cég egyik partnere. Írásai a Dr. Dobb's Journalban, a C++ Reportban, a C/C++ Users Journalban, a Data Based Advisorban jelentek meg, és számos tanácskozás anyagában megtalálhatók.

Kilenc SSH-trükk

Titkosítsuk minden kapcsolatunkat SSH segítségével!

Az SSH az `rsh` és az `rlogin` távoli bejelentkezésre használható, de nem titkosított programok utódja. Az `rsh` és az `rlogin`, akárcsak a `telnet`, régi származással büszkélkedhet, mostanra azonban túlhaladottá és biztonsági szempontból elégtelenné vált. Ugyanakkor e programok meglepően nagyszámú, igen hasznos szolgáltatást gyűjtöttek össze a Unix-fejlesztések két évtizede alatt, így a legjobbak közülük az SSH-ba is bekerültek. Következzék az a 9 trükk, amit a leghasznosabbnak találtam. Hozzuk ki a legtöbbet az SSH programból! Ez a cikk is az OpenSSH-n alapul, ezért ha esetleg valamilyen más változatot használnak, a trükkök kipróbálása előtt ellenőrizze a leírást.

X11-átirányítás

Az SSH-n keresztül X-es kapcsolatainkat titkosíthatjuk. Ráadásul nemcsak a forgalom lesz titkosított, hanem a távoli gép `DISPLAY` környezeti változója is a megfelelő értéket veszi fel. Ha tehát a helyi gépünkön X-et futtatunk, távoli X-alkalmazásaink varázslatos módon a mi képernyőnkön jelennek majd meg.
Az X11-átirányítást az `ssh -X` *gépnév* paranccsal kapcsolhatjuk be. Az X11-átirányítást csak olyan gépekről használjuk, amelyeknek a rendszergazdájában megbízunk, ellenkező esetben az X11 alapú támadásokkal szemben sebezhetőkké válunk. Egy, az X11-átirányításon alapuló ügyes trükkel képeket jeleníthetünk meg az `xterm` ablakokban. Futtassuk a beépített képnézővel ellátott `w3m` böngészőt a távoli gépen, figyeljük meg a `w3m-img` Debian csomagot vagy a `w3m-imgdisplay` RPM-et. Ezek az X11-átirányítás segítségével `xterm` ablakunk felett egy keret nélküli ablakot nyitnak. Ha levelünket szöveges ügyfél segítségével, távolról olvassuk SSH-n keresztül, ezzel a módszerrel ugyanabban az `xterm` ablakban megnézhetjük a benne rejlő képességeket.

Beállításfájl

Az SSH a ***~/.ssh/config*** helyen keresi a felhasználói beállításfájlt. Ez például a következőképpen nézhet ki:

```
ForwardX11 yes
Protocol 2,1
```

A `ForwardX11 yes` ugyanazt jelenti, mintha a `-X` kapcsolót adtuk volna meg a parancssorban. A `Protocol` sor adja meg az SSH-nak, hogy előbb az SSH2 rendszert használja, és ha nem megy, csak akkor lépjen vissza a SSH1-hez. Amennyiben kizárólag SSH2-t használunk, töröljük a `,1` részt.
A beállításfájlunkban a `Host` utasítás alkalmazásával szakaszokat is kialakíthatunk, amelyek csak bizonyos távoli gépekhez való kapcsolódás során lépnek életbe. Hasznos beállításfájl-utasítás a `User`, amely a távoli gépen adja meg felhasználónevünket. Ha `ssh -l távolifelhasználó távoligép` vagy `ssh távolifelhasználó@távoligép` alakban gyakran jelentkezünk be valamelyik gépre, érdemes a beállításfájlunkba illeszteni a következő sorokat:

```
Host távoligép
ForwardX11 yes
User távolifelhasználó

Host *
ForwardX11 no
```

Mostantól csak a távoli gépet kell begépelnünk, ha a ***ForwardX11*** képesség bekapcsolásával távoli felhasználóként akarunk bejelentkezni. Minden más esetben a ***ForwardX11*** legyen kikapcsolva, ahogy azt fentebb ajánlottuk. A csillag minden gépnévvel egyezőnek tekintendő, még azokkal a nevekkel is, amelyeket korábban a `Host` szakaszokban felsoroltunk, de mindig csak az első találat utasítását használja a rendszer. Az egyedi `Host` szakaszokat ezért beállításfájlunkban az általános `Host` szakasz elé tegyük.
Az SSH rendszerbeállításfájllal is rendelkezik, amelynek neve ***/etc/ssh/ssh_config***. Az SSH a következő sorrendben gyűjti a beállításadatokat: parancssori kapcsolók, felhasználói beállításfájl, végül rendszer-beállításfájl. Az összes kapcsolót megtalálhatjuk a man `ssh_config` szövegében.

Gyorsítsunk: tömörítés és kódolások

Az SSH bármelyik kapcsolaton képes használni a `gzip` tömörítést. Az alapértelmezett tömörítés körülbelül 4×-es szövegtömörítésnek felel meg. A tömörítés különösen jól jöhet, ha például X-alkalmazást szeretnénk modemes vagy lassú hálózati kapcsolaton átirányítani. A tömörítést az `ssh -C` kapcsolóval vagy a ***Compression yes*** beállításfájlba illesztésével kapcsolhatjuk be.
Egy másik sebességgyorsító módosítás lehet titkosító kódolásunk megváltoztatása. Számos régi rendszer alapértelmezett kódolása a háromszoros DES (3DES), amely lassabb a Blowfish vagy AES kódolásnál. Az OpenSSH új változatai alapértelmezés szerint a Blowfisht használják. A kódolást az `ssh -c blowfish` kapcsolóval állíthatjuk át.
Beállításfájlunkban attól függően kell beállítanunk a kódolást, hogy SSH1 vagy SSH2 alapú rendszert használunk-e. Az SSH1 esetében a `Cipher blowfish`, míg SSH2 alatt a `Ciphers blowfish-cbc,aes128-cbc,3des-cbc,cast128-cbc,arcfour,aes192-cbc,aes256-cbc` alakot használjuk.

Kapuátirányítás

A kapuk a kiszolgáló különféle szolgáltatásait jelképező számok; például a 80-as kapu a HTTP-szolgáltatásnak, míg a 110-es kapu a POP3-nak felel meg. A szabványos kapuszámokhoz tartozó szolgáltatások listáját a ***/etc/services*** állományban találjuk. Az SSH gépünk bármelyik kapujáról képes adatokat átküldeni egy SSH-t futtató távoli gépre, s a forgalmat aztán a távoli kiszolgáló a másik gép tetszőleges kapujára irányíthatja. De miért akarnánk mi ilyesmit? Két okból: a titkosítás kedvéért és a csövezett (tunelling) kapcsolatért.

Titkosítás

Egy csomó alkalmazás használ olyan protokollt, amelyben a jelszavak és az adatok egyszerű szövegként utaznak a hálózaton. Ilyen a POP3, IMAP, SMTP és az NNTP. Az SSH átlátszó módon képes őket titkosítani. Tegyük fel, hogy a levelezőprogramunk normál esetben a ***mail.example.net*** POP3-kapujára szokott kapcsolódni (110-es kapu). Továbbá tegyük fel, hogy a ***mail.example.net*** gépre ugyan az SSH-val nem tudunk közvetlenül belépni, viszont van azonosítónk a ***shell.example.net***-en. Ilyenkor megmondhatjuk az SSH-nak, hogy titkosítsa a saját gépünk 9110-es kapuját (a számot tetszőlegesen választhatjuk meg), és a ***shell.example.net*** SSH-kiszolgálóját felhasználva küldje a ***mail.example.net*** 110-es kapujára:

```
ssh -L 9110:mail.example.net:
➥110 shell.example.net
```

Azaz a helyi 9110-es kaput továbbítsd a ***mail.example.net*** 110-es kapujára a ***shell.example.net***-re nyitott SSH-kapcsolaton keresztül. Ezután nincs más dolgunk, mint a levelezőprogramunknak megadni, hogy a helyi gépünk 9110-es kapujához kapcsolódjon. Itt az adat titkosítódik, az SSH-kapun keresztül átkerül a ***shell.example.net***-re, ahol visszakódolódik, végül a ***mail.example.net*** 110-es kapujára továbbítódik. Hasznos mellékhatásként a ***mail.example.net*** POP3-démonja azt fogja hinni, hogy minden forgalmat a ***shell.example.net***-től kap.

Csövezett kapcsolatok

Az SSH képes a tűzfalon átívelő hídként működni – védje az a tűzfal akár a saját gépünket, akár a távoli gépet, netalán mindkettőt. Az egyetlen dolog, amire szükségünk van, egy ismert SSH-kiszolgáló a tűzfal másik oldalán. Például igen sok DSL és kábelmodemes cég tiltja a levelek küldését saját gépünk 25-ös (SMTP) kapuján keresztül.
Következő példánkban kábelmodemes kapcsolatunkon keresztül levelet küldünk cégünk SMTP-kiszolgálójáról. A példában felhasználtuk a ***mail.example.net*** nevű SMTP-kiszolgálón meglévő azonosítónkat. Az SSH-parancs a következő lesz:

```
ssh -L 9025:mail.example.net:
➥25 mail.example.net
```

Ezt követően levélküldő programunknak mondjuk meg, hogy a helyi gép 9025-ös kapuját használja levélküldésre. Ez a gyakorlat igen hasonlónak tűnik az előzőhöz, csak most a ***mail.example.net*** 25-ös kapujára a ***mail.example.net***-en keresztül a helyi 9025-ös kapuról nyitottunk csatornát. A tűzfal szemszögéből mindössze szabályos SSH-adatok utaznak a szabványos SSH-kapun (22) keresztül, köztünk és a ***mail.example.net*** között.

Bináris adatok csövezése távoli héjprogramba

Az SSH-n keresztül használt csövezés távoli héjprogramokba teljesen átlátszó módon működik. Figyeljük meg a következőket:

```
cat myfile | ssh user@desktop lpr
```

```
tar -cf - source_dir | ssh user@desktop
➥'cat > dest.tar'
```

Az első példa a ***myfile*** állományt csövezi az a ***desktop*** nevű gépen futó `lpr` programba. A második példa egy tarfájlt hoz létre, és a tartalmát a terminálra írja (mivel a tarfájl nevének helyére egy kötőjelet írtunk), amit aztán átcsövezünk a ***desktop*** nevű gépre, és átirányítjuk egy fájlba.

Távoli héjparancsok futtatása

Az SSH segítségével nem feltétlenül kell felhasználói beavatkozással (interactive) héjprogramot nyitnunk, ha mindössze csak a távoli program kimenetére vagyunk kíváncsiak. Például a következőt is megtehetjük:

```
ssh felhasználó@gép w
```

A fenti parancs felhasználóként lefuttatja a `w` parancsot a ***gép*** nevű gépen, majd megjeleníti az eredményt. Használhatjuk a parancsok önműködővé tételére is:

```
perl -e 'foreach $i (1 .. 12) \
{print `ssh server$i "w"`}'
```

Ügyeljünk az SSH-parancs körüli fordított egyszeres idézőjelekre (aposztróf). A fenti sor Perl segítségével 12 alkalommal hívja meg az SSH-t, mindannyiszor más-más távoli gépen futtatva le a `w` parancsot, kezdve a ***server1***-től egészen a ***server12***-ig. De minden egyes SSH-kapcsolat létrejöttekor be kell majd gépelnünk a jelszavunkat. Ha viszont továbbolvasunk egy kicsit, megtudhatjuk, hogyan kerülhetjük el a biztonság feláldozása nélkül a jelszógépelést. (Ugyanezt sokkal egyszerűbben és erőforrás-takarékosabban az alábbi héjprogaramocska is tudja:

```
for i in `seq 1 12` ; do ssh server$i "w"; done
```

– ***Mészáros Gergely***, a fordító.

Utazzunk SSH Java Applettel

Seregnyi ember szaladgál lemezen magával hordozva a PuTTY-ot vagy valamilyen hasonló windowsos SSH-programot, hátha utazásai során nem biztonságos gépet kell használnia. Ez a módszer azonban csak akkor működik, ha lemezről lehet futtatni a programokat. Igaz, a PuTTY programot a honlapjáról is letölthetjük és elindíthatjuk.
Egy másik lehetséges megoldás, ha az előzőleg valamely honlapra felrakott SSH Java appletet böngészőből használjuk. Például kitűnő Java SSH-ügyfél a Mindterm, ennek nem üzleti célú felhasználása ingyenes. A programot a
➲ http://www.appgate.com/mindterm oldalon találjuk meg.

Összegzés

Az SSH beállítása a trükkök alkalmazása során néhány helyen könnyen félresiklik. Számos hibát felderíthetünk az `ssh -v` kapcsolóval és a kimenet figyelésével. Kétségtelen, hogy a trükkök egyike sem nélkülözhetetlen az SSH használatához, viszont könnyen kerülhetünk olyan helyzetbe, amikor örülünk, hogy mégis ismerjük őket. Nem árthat meg, ha kipróbálunk közülük néhányat!

Linux Journal 2003. augusztus, 112. szám

Daniel R. Allen (da@coder.com)
1995 óta elfogult Linux-rajongó. Egy kitcheneri programtanácsadó cég, a Prescient Code Solutions elnöke Ontario és Ithaca, területén.

VTun

Kapcsoljuk össze otthonunkat és munkahelyünket virtuális magánhálózaton keresztül!

A dotcom-korszak boldog, szép napjaiban egy P2P-programot fejlesztő, induló cég legelső alkalmazottja voltam. Mivel az intranetet és a fejlesztőkörnyezetet az alapjaitól kellett felépítenünk, mindenhol használhattuk a Linuxot. Mint tudjuk, a világ megváltozott, a dotcomok a dodó madár sorsára jutottak. Induló vállalkozásom is így járt, felvásárolta egy nagyobb vállalat, amely Windows-alapon fejlesztett. Bár az új cég elég engedékeny volt, így továbbra is Linuxszal dolgozhattam, de az ezzel kapcsolatos rendszergazdai feladatok is teljesen rám hárultak.

Egyetlen terület volt, ahol komoly nehézséggel kellett szembenéznem: a VPN beállítása. A régi munkahelyemen minden fejlesztőnek volt bemenő SSH-kapcsolata a saját munkaállomására. Az új munkahelyemen nemcsak az SSH-kapukat zárták le, de a rendszeresített VPN-megoldás sem volt Linux-barát. A tehetetlenség törvényéből fakadóan várható volt, hogy a több felületen is működőképes megoldások, például a FreeS/WAN, nem fognak egyhamar bevezetésre kerülni. Szerencsére a VTun, a régi munkahelyemen használt VPN-megoldás elég rugalmas volt ahhoz, hogy ebben a barátságtalan környezetben is helytálljon.

Hogyan működik?

A VTun úgy működik, hogy az IP-alagutazást észrevétlenül a meglévő csomagtovábbító programokon keresztül valósítja meg. A unixos moduláris szemléletnek megfelelően a VTun közvetlenül csak a csomagok alagutaztatásáért felel a két rendszer között, és a meglévő hálózati segédeszközöket használja a teljes VPN-megoldás kialakítására. A hasonlat kedvéért képzeljük el, hogy az otthoni és a munkahelyi hálózat két különálló vasúti hálózat. Minden számítógépnek egy állomás felel meg. A Linux-rendszermag vezérli a váltókat, így határozza meg, hogyan érik el a vonatok egyik állomásról a másikat. Ezeket a lehetőségeket a `route` programmal befolyásolhatjuk, így a végfelhasználó is adhat hozzá vagy távolíthat el állomásokat. A Linux-rendszermag képes a vonatok útvonalát megváltoztatni. Vasutaspéldánkhoz adjuk hozzá az internetet, a hatalmas vasúthálózatot. Az otthoni és a munkahelyi hálózatok aprócska nyúlványok ebben a rendszerben. Általában egyetlen állomás, a tűzfal vagy az átjáró rendelkezik közvetlen hozzáféréssel az internet vasúthálózatához. Ha egy másik állomás az otthoni vágányokról vonatot akar küldeni az internetre, először az átjáróállomásra kell továbbítania. Ezt az útvonal-módosítást, ami műszakilag az IP-álcázásnak vagy a hálózati címfordításnak felel meg, az `iptables` program vezérli. Az `iptables` a 2.4-es Linux-rendszermagban lévő Netfilter tűzfalkód felhasználói térben futó fele.

Hogyan illeszthető be a VTun a példába? Emlékezzünk, hogy az otthoni és a munkahelyi hálózatok egymástól elszigetelt vasúthálózatok. Otthonról általában nem mehet vonat a munkahelyre, mert a munkahelyi tűzfalállomás nem engedi át. A VTun segítségével virtuális sínpárt építhetünk két elkülönült hálózaton lévő állomás – például egy otthoni és egy munkahelyi gép – között. A sínpár lefektetése után az állomásokon be kell állítani az `iptables`-t és a `routed`-et, hogy az otthonról érkező vonatok szabadon elérhessék a munkahelyi rendszert, mintha csak egy munkahelyi gépről érkeztek volna.

Szabályok és kikötések

Most, hogy áttekintettük a VTun VPN-összetevőit, készen állunk a teljes megvalósítás vizsgálatára. A leghétköznapibb eset, amikor egyetlen távoli munkaállomást (az otthoni gépünket) kötjük össze a munkahelyi belső hálózattal a munkahelyi gépünkön keresztül. Az egyszerűség kedvéért feltesszük, hogy lehetséges SSH-kapcsolatot létesíteni otthonról a munkahelyi géppel, de az a gép egyébként nem érhető el az internetről. Tegyük fel, hogy az otthoni hálózat a 192.168.1.0/24 alhálózatra, a munkahelyi hálózat a 192.168.5.0/24 és a 192.168.100.0/24 alhálózatokra van beállítva.

A VTun kiszolgálóalapú rendszer. A kiszolgálógép a megadott kapukon figyeli a bejövő kapcsolatokat. Az ügyfél kezdeményezi az alagút létrehozását a kiszolgáló kapujára csatlakozva – ebben a példában az otthoni gép az ügyfél és a munkahelyi gép a kiszolgáló.

A VPN létrehozása azt jelenti, hogy a munkahelyi hálózat onnantól fogva csak annyira biztonságos, mint az otthoni hálózat. Emiatt az otthoni gépeket kötelező tűzfallal védeni, amire mindig fel kell rakni az összes biztonsági foltot, és behatolásvédelmi szempontból rendszeresen ellenőrizni kell. A másik nagyon fontos szabály, hogy soha ne hozzunk létre VPN-t a munkahelyi rendszergazdák tudta és beleegyezése nélkül.

Telepítés

A VTun programot az ügyfélre és a kiszolgálóra is telepíteni kell, azaz az itt leírt folyamatot mindkét rendszeren végre kell hajtani. A folyamatot a Red Hat Linux újabb változatain próbálták ki. Ha a telepítés nem sikerül a terjesztéseden, küldj levelet a ryan@ryanbreen.com címre. Ezeket a válaszokat beépítem a ➲ http://www.ryanbreen.com/vtun honlapon fenntartott terjesztésfüggő hibákat felsoroló állományba. Néhány terjesztés eleve tartalmazza a VTun-csomagot, úgyhogy elképzelhető, hogy megtakaríthatunk egy lépést, ha a csomagkezelővel telepítjük a VTun programot.

A legtöbb VPN-megoldáshoz hasonlóan a VTun is igényli a rendszermag támogatását, ebben az esetben a TUN pont–pont hálózati illesztőprogramot. A `TUN` modul része az alaprendszermagnak, úgyhogy valószínűleg nem kell rendszermagot fordítani hozzá. Tegyünk egy próbát az `insmod tun` parancscsal (rendszergazdaként), amely megkísérli az illesztőprogram betöltését. Ha a modul nem található, töltsük le a legújabb változatot (jelenleg ez a tun-1.1) a ➲ http://vtun.sourceforge.net/tun/index.html címről. Telepítsük:

```
tar xzf tun-1.1.tar.gz
cd tun-1.1
su -c ´make install´
```

Ha önműködően szeretnénk betölteni a TUN modult, amikor egy folyamat megkísérli elérni a virtuális alagúteszközt, akkor a következő sort adjuk a */etc/modules.conf* állományhoz:

```
alias char-major-10-200 tun
```

Ezután állítsuk be és telepítsük a felhasználói térben futó vtund programot. A legújabb VTun-csomag a ➲ http://vtun.sourceforge.net/download.html oldalról érhető el. Most forrásból telepítünk, de ha a terjesztésünk támogatja az RPM- vagy DEB-csomagokat, akkor nyugodtan telepítsük az előre lefordított változatot. A legújabb forráscsomag a cikk nyomdába adásának pillanatában a vtun-2.5.tar.gz.
A fordítás a szokásos módon zajlik:

```
tar xzf vtun-2.5.tar.gz
cd vtun-2.5
./configure
make
su -c ´make install´
```

Egyes terjesztéseknél a beállítólépés sikertelen lehet, mert az *LZO* nincs telepítve. Az *LZO* egy tömörítő programkönyvtár, amelyet a VTun használ.
A ➲ http://www.oberhumer.com/opensource/lzo/download weboldalról tölthető le. Fordítsuk le és telepítsük az *LZO*-t, majd próbáljuk újra a VTun telepítését.
A telepítés során a VTun a beállítóállományát a */usr/local/etc/vtund.conf* helyre teszi. Ez nagyon zavaró lehet, mert az ügyfélnek és a kiszolgálónak mást-mást kell beállítani az alagútleíró részben. A félreértések elkerülése érdekében a *vtund.conf* állományt nevezzük át *vtund-client.conf*, illetve *vtund-server.conf* névre. Ezt követően az indításkor kézzel adjuk meg a megfelelő beállítóállomány elérési útját. A következőkben e javaslat szerint járunk el.

A VTun beállítóállománya

A VTun beállítóállománya viszonylag egyszerű (lásd az *1.* és a 2. listát – az utóbbi megtalálható az 51. CD Magazin/ VTun könyvtárában). Az állomány három különálló részből áll. Az elsőben általános beállítások szerepelnek, például a kiszolgáló kapuszáma és a segédprogramok elérési útja. A másodikban az alapértelmezett munkamenet beállításai vannak, amelyek az alagút hálózati tulajdonságait írják le. Ezeket a tulajdonságokat szükség esetén egy adott alagút beállításánál felül lehet bírálni.
Van egy alagútbeállító kapcsoló, amely különleges figyelmet igényel: ez a keepalive. A vállalati rendszergazdák gyakran alacsony tétlenségi időt engedélyeznek a tűzfalon keresztül a működő kapcsolatok számára. Ha az alagút a megadott időnél tovább tétlen, a kapcsolat időtúllépéssel megszakad. A keepalive engedélyezésével a VTun úgy kerüli meg ezt a gondot, hogy rendszeresen csomagokat küld az ügyféltől a kiszolgálóra, így meggyőzi a tűzfalat arról, hogy a kapcsolat él.
A beállítások utolsó csoportja az adott alagút beállításait tartalmazza. A beállítóállomány tetszőleges számú ilyen beállítást tartalmazhat, így több VPN kialakítása is lehetővé válik az ügyfelek és a kiszolgálók számára. Minden alagútbeállító csoport egy névvel kezdődik. Én a my_tunnel nevet választottam, de bármit megadhatunk. Minden alagút jelszóval védhető, de ezzel általában nem foglalkozunk, ha az alagút SSH-n keresztül jön létre. Az up és down részek az alagút felépülésekor, illetve lebontásakor végrehajtandó parancsokat tartalmazzák.
Az 1. és 2. listán látható egyszerű beállítóállományok arra utasítják a VTun programot, hogy kapcsolatfelvételkor hozza létre az alagút csatolóit. A beállítóállományokban a %% minta jelenti az alagútcsatolót, így több alagutat is létrehozhatunk tetszőleges sorrendben. Az alagútcsatoló tényleges neve a „tun” előtagból és egy számjegyből áll. Az első létrehozott alagút neve tun0.

1. lista Egyszerű vtund-client.conf

```
options {
    port 5000;

    # Különféle programok elérési útja
    ifconfig    /sbin/ifconfig;
}

# Munkamenet alapértékei
default {
    compress no;    # Nincs tömörítés
    encrypt no;     # Az SSH úgyis titkosít
    speed 0;        # Legnagyobb sebesség
alapértelmezettként
    keepalive yes;
    stat yes;
}


my_tunnel {
    pass   XXXXXXXX;     # Jelszó
    type   tun;           # IP-alagút
    proto tcp;            # TCP protokoll

up {
    # 10.3.0.1 = hamis alagútcsatoló (otthon)
    # 10.3.0.2 = hamis alagútcsatoló
                ➥(munkahelyen)
    # 192.168.5.0/24 = az 1. munkahelyi
                      ➥hálózat
    # 192.168.100.0/24 = a 2. munkahelyi
                       ➥hálózat
    ifconfig
      "%% 10.3.0.1 pointopoint 10.3.0.2 mtu
       ➥1450";
};
down{
    ifconfig "%% down";
};
}
```

VTun VPN létrehozása

Próbáljuk ki a gyakorlatban a VTun beállításáról tanultakat. Az 1. és 2. lista felhasználásával hozzunk létre egy egyszerű alagutat. A listák az ➲ ftp.ssc.com/pub/lj/listings/issue112/6675.tgz címen megtalálhatók, ha nem szeretnénk begépelni őket. Mentsük a *vtund-server.conf* állományt a munkahelyi gép */usr/local/etc* könyvtárába, illetve a *vtund-client.conf* állományt az otthoni gép */usr/local/etc* könyvtárába. Miután a beállítóállományok a helyükre kerültek, mindkét gépen indítsuk el a VTun-folyamatokat. Rendszergazdaként indítsuk el a kiszolgálót a munkahelyi gépen:

```
vtund -f /usr/local/etc/vtund-server.conf -s
```

3. lista Teljes vtund-client.conf

```
options {
    port 5000;

    # Különféle programok elérési útja
    ifconfig    /sbin/ifconfig;
    firewall    /sbin/iptables;
    route       /sbin/route;
}

# Munkamenet alapértékei
default {
    compress no;    # Nincs tömörítés
    encrypt no;     # Az SSH úgyis titkosít
    speed 0;        # Legnagyobb sebesség
                    # alapértelmezettként
    keepalive yes;
    stat yes;
}

my_tunnel {
    pass  XXXXXXXX;   # Jelszó
    type  tun;        # IP-alagút
    proto tcp;        # TCP protokoll

up {
    # 10.3.0.1 =  hamis alagútcsatoló
                  ➥(otthon)
    # 10.3.0.2 =  hamis alagútcsatoló
                  ➥(munkahelyen)
    # 192.168.5.0/24 = az 1. munkahelyi
                       ➥hálózat
    # 192.168.100.0/24 = a 2. munkahelyi
                         ➥hálózat
    ifconfig
      "%% 10.3.0.1 pointopoint 10.3.0.2 mtu
       ➥1450";
    route "add -net 192.168.5.0 netmask
    ➥255.255.255.0 gw 10.3.0.2";
    route "add -net 192.168.100.0 netmask
    ➥255.255.255.0 gw 10.3.0.2";
};
down{
    ifconfig "%% down";
    route "del -net 192.168.5.0 netmask
    ➥255.255.255.0 gw 10.3.0.2";
    route "del -net 192.168.100.0 netmask
    ➥255.255.255.0 gw 10.3.0.2";
    };
}
```

A `-s` kapcsoló hatására a `vtund` kiszolgálóként indul, a kapcsolatokra az 5000-es kapun vár. A kiszolgáló eléréséhez el kell tudnunk érni az 5000-es kaput a munkahelyi gépen. Emlékezzünk rá, hogy példánkban feltettük, hogy a munkahelyi gép csak SSH-n keresztül érhető el, ezért az SSH kaputovábbító képességét kell használnunk a munkahelyi gép 5000-es kapujához való alagutazásra. Otthonról adjuk ki a következő parancsot:

```
ssh mydesktop.work.com -L 5000:localhost:5000
```

A `-L` kapcsoló hatására az OpenSSH az otthoni gép 5000-es kapuját a munkahelyi gép 5000-es kapujára irányítja. Az otthoni gép 5000-es kapujára irányított kapcsolatok SSH-n keresztül észrevétlenül átalagutaznak a munkahelyi gép 5000-es kapujára. Az elrendezésnek további előnye az, hogy a VPN-es forgalom titkosítva van.
Miután a munkahelyi gépen futó kiszolgáló elérhető az otthoni gépről, csak az marad hátra, hogy elindítsuk az ügyfelet. Rendszergazdaként az otthoni gépen futtassuk a következő parancsot:

```
vtund -f /usr/local/etc/vtund-client.conf
my_tunnel localhost
```

A `my_tunnel` kapcsoló megadja az ügyfélnek és a kiszolgálónak, hogy milyen alagutat kell létrehozni. Mindkét rendszer lekéri és futtatja a megfelelő beállítóállományokból a `my_tunnel` részhez tartozó `up` bejegyzésben megadott parancsokat. Az utolsó kapcsoló, a `localhost`, megadja a VTun kiszolgáló gépnevét. Ebben az esetben a VTun kiszolgáló a `localhost`, mert az 5000-es kaput átirányítottuk az otthoni gépről a munkahelyi gépre.
Ha az alagút sikeresen létrejött, akkor mindkét gépen megjelenik az `ifconfig` kimenetében a `tun0` csatoló. Az otthoni gépnek az IP-címe 10.3.0.1 a `tun0`-n, a munkahelyi gépnek az IP-címe 10.3.0.2. A vonatos példára visszatérve, létrejött egy sínpár az otthoni és a munkahelyi gépek között, és most már irányíthatunk rá vonatokat. Bemutatásként hozzunk létre egy SSH-kapcsolatot az otthoni gépről a 10.3.0.2-re.

Beüzemelés

Van már egy működő alagutunk otthonról a munkahelyre. A következő lépésben a `route` és `iptables` programok segítségével be kell állítanunk, hogy az otthonról jövő csomagok a munkahelyi gépen keresztül álcázódjanak a munkahelyi hálózat felé. Szerencsére ez egyszerűen megtehető, csak egy pár sort kell hozzáadni beállítóállományokhoz az ügyfélen és a kiszolgálón, és újra kell indítani a `vtund` folyamatokat.
A VTun a kapcsolat létrejöttekor végrehajtja a megfelelő `route` és `iptables` parancsokat.
A vonatos példával elmagyarázva ez azt jelenti, hogy az otthoni állomásról minden munkahelyi hálózatra irányuló vonatot az újonnan létrehozott VTun-sínpáron keresztül kell küldeni. Kézzel ez így állítható be:

```
route add -net 192.168.5.0 netmask
➥255.255.255.0 gw 10.3.0.2
route add -net 192.168.100.0 netmask
➥255.255.255.0 gw 10.3.0.
```

A másik megoldás, hogy a *3. listán* látható módon adjuk hozzá a parancsokat a ***vtund-client.conf*** állományhoz. A parancsok hatására az `iptables` minden csomagot továbbít a `tun` csatolóról, és úgy álcázza őket, mintha a munkahelyi gépről indulnának. Megtehetjük azt is, hogy a *4. listán* látható parancsokat adjuk hozzá a ***vtund-server.conf*** állományhoz, és újraindítjuk a kiszolgálót.
A `route` és az `iptables` beállítása után az egész vállalati belső hálózat látszani fog otthonról. Böngészhetjük a belső webkiszolgálókat, kapcsolódhatunk a forráskód-kiszolgálóhoz, és megpróbálhatunk grafikus elemeket (például egy `xterm`-et) exportálni. Ezekre a célokra több mint elegendő a teljesítmény, és az SSH-alagút biztosítja, hogy a forgalom a kíváncsi szemek elől rejtve marad.
Ha a kiszolgálót esetleg önműködően szeretnénk elindítani,

4. lista Teljes vtund-server.conf

```
options {
    port 5000;

    # Különféle programok elérési útja
    ifconfig    /sbin/ifconfig;
    firewall    /sbin/iptables;
    route       /sbin/route;
}

# Munkamenet alapértékei
default {
    compress no;    # Nincs tömörítés
    encrypt no;     # Az SSH úgyis titkosít
    speed 0;        # Legnagyobb sebesség
                    # alapértelmezettként
    keepalive yes;
    stat yes;
}

my_tunnel {
    pass  XXXXXXXX;    # Jelszó
    type  tun;         # IP-alagút
    proto tcp;         # TCP protokoll

up {
    # 10.3.0.1 = hamis alagútcsatoló (otthon)
    # 10.3.0.2 = hamis alagútcsatoló
                 ➥(munkahelyen)
    # 192.168.1.0/24 = az otthoni hálózat
    ifconfig
      "%% 10.3.0.2 pointopoint 10.3.0.1 mtu
       ➥1450";
    route "add -net 192.168.1.0 netmask
    ➥255.255.255.0 gw 10.3.0.1";
    firewall "-t nat-A POSTROUTING -o %%
    ➥-j MASQUERADE";
    firewall "-AFORWARD -i %% -j ACCEPT";
};
down{
    ifconfig "%% down";
    route "del -net 192.168.1.0 netmask
    ➥255.255.255.0 gw 10.3.0.1";
};
}
```

akkor a használt terjesztéstől függ a teendőnk. A VTun-forráscsomag tartalmaz néhány indítóparancsfájlt a különböző terjesztésekhez – olvassuk el a ***Readme*** állományt, majd válasszuk ki a legmegfelelőbbet.

Beállítások haladóknak

Észrevehettük, hogy csak egyetlen otthoni gépnek volt hozzáférése a munkahelyi intranethez. Az otthoni hálózat más állomásairól induló vonatok nincsenek átirányítva. Ez a megoldás egy hajszálnyival biztonságosabb, mert csökkenti a munkahelyi hálózatot fenyegető, az otthoni hálózatról eredő hibákból adódó veszélyeket. Ha az otthoni hálózat más gépeiről is el szeretnénk érni a munkahelyi hálózatot, akkor egyszerűen adjuk hozzá a megfelelő `iptables`-szabályokat a ***vtund-client.conf*** állomány `up` részéhez. Ezt az érdeklődő olvasó gyakorlásképpen elvégezheti.

A fenti megoldás tökéletesen működik, ha bármelyik munkahelyi gép elérhető SSH-n keresztül. Sajnos sok munkahelyen egyáltalán nem hagynak nyitva egy bejövő kaput sem. Pontosan ez volt a helyzet az új munkahelyemen is, de a rugalmassága átsegített ezen az akadályon is. A megoldás a szerepek felcserélése: a munkahelyi gép lesz az ügyfél, és az SSH-alagutat a munkahelyről kezdeményezzük.

A megoldás akkor működőképes, ha az otthoni gépet a munkahelyünkről el tudjuk érni. A legtöbb széles sávú szolgáltató viszont dinamikus IP-címet oszt. A gond a dinamikus IP-kre kifejlesztett DNS-szolgáltatások egyikével áthidalható, például a Dyndns.org által nyújtottal.

A legnagyobb hátrány a viszonylagos törékenység. Biztonságos környezetben az ügyfél nem indul el önműködően, mert az SSH-kapcsolathoz hitelesítés szükséges. Egy munkahelyi áramszünet miatt könnyen kizáródhatunk. Ha kevésbé aggódunk a biztonság miatt, önműködővé tehetjük a bejelentkezést az SSH nyilvános kulcsos hitelesítése segítségével, ha nincs a kulcsnak jelszava, illetve az `expect` program parancsállományba építésével. Egyik módszert sem javaslom.

Ha a munkahelyi gép szünetmentes tápegységre van kötve, akkor ritkán jön elő ez a gond. Az elmúlt hat hónapban, amióta ezt a megoldást használom, csak egyszer tartott olyan hosszú ideig az áramszünet, hogy a VPN-em ügyféloldala meghaljon tőle. A megoldás az otthoni hálózat oldalán is megbízható. A számítógép napokra is hálózati kapcsolat nélkül maradhat, a VPN újraindul, ahogy elindítjuk a `vtun` kiszolgálót, hála az ügyfél okos életbentartó és újrapróbálkozó képességeinek.

Összefoglalás

A VTun képességeinek teljes ismertetése jóval meghaladná e cikk kereteit. Az itt bemutatott egyszerű megoldásokon kívül a VTun lehetővé teszi IP-n kívüli protokollok alagutaztatását is etherneten, PPP-n vagy SLIP-en keresztül. A VTun saját maga is képes titkosításra, tömörítésre és sávszélesség-formázásra, úgyhogy minden elképzelhető összeköttetéshez alkalmazható.

Köszönetnyilvánítás

Nagyon köszönöm ***Jennifer Edwardsnak*** és ***James Manningnak***, hogy lektorálták a cikket.

A cikkhez kapcsolódó listák megtalálhatóak az 51. CD Magazin/VTun könyvtárában.

Linux Journal 2003. augusztus, 112. szám

Ryan Breen (ryan@ryanbreen.com)
A Duke Egyetemen szerzett számítástechnikai és közgazdasági diplomát. Bostonban él a barátnőjével és a kutyájával. Nagy áteresztőképességű böngészőszimulációkat épít, elkötelezett KDE-felhasználó, és néha fejleszt is KDE alá.

KAPCSOLÓDÓ CÍMEK

VTun ➲ http://vtun.sourceforge.net
Ryan VTun lapja ➲ http://www.ryanbreen.com/vtun
VTun TUN/TAP meghajtó ➲ http://vttun.sourceforge.net/tun

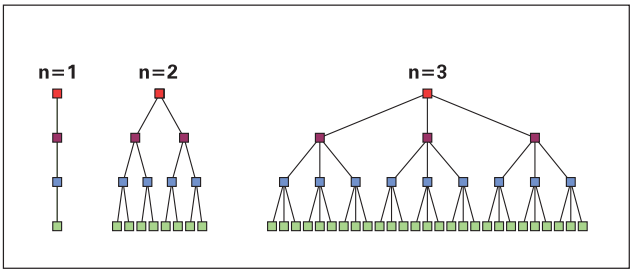
Reiser4: hatékonyan gyorstárazó fák tervezése (2. rész)

A Reiser fájlrendszer negyedik változata a nagyobb teljesítmény érdekében jobban ellapítja a faadatszerkezetet, mint a harmadik változat tette. A leírtakból megtudhatjuk, hogyan is fest ez a gyakorlatban.

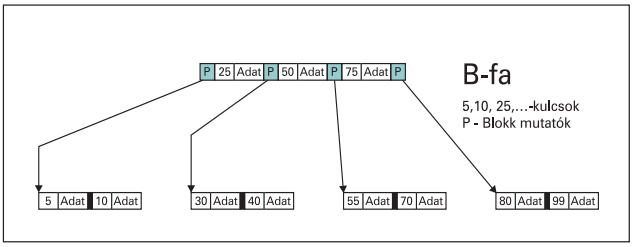
Cikkünk a Reiser fájlrendszer felépítéséről szóló cikksorozatunk második része. Az első cikkben (Linuxvilág, 2003. március) az alapokat tárgyaltuk: a fákat, a csomópontokat és a cikkelyeket. Ebből a cikkből azt tudhatod meg, hogy miért jobbak a kiegyensúlyozott fák a kiegyensúlyozatlan fáknál, és miért jobbak a B+ fák a B-fáknál, miközben a gyorstárazási alapelveket is megismered. Ezt követően arra is fény derül, hogyan alkalmazza ezeket az említett elemeket a Reiser fájlrendszer az adatbázis-rendszereknél már megismert BLOB-ok, vagyis nagyméretű bináris egységek esetében. Mindez azt sugallja, hogy a BLOB-ok csökkentik a belső csomópontok gyorstárazásának a hatékonyságát, mivel a fa így kiegyensúlyozatlanná válik. Azt is megtudjuk, hogyan tárol a Reiser fájlrendszer olyan szerkezeteket, amelyek nagyobb méretűek egy csomópontnál, téve ezt anélkül, hogy a fa kiegyensúlyozatlanná válna.
Elnézést az olvasóktól, hogy mindenkit ennyire megvárattam ezzel a cikkel – a késedelem oka a Halloweenkor esedékes 2.6 miatti bővítésbefagyasztás volt, mivel akkortájt kellett a Reiser4-et gyorsan megbízható formába hozni.

Elágaztatás

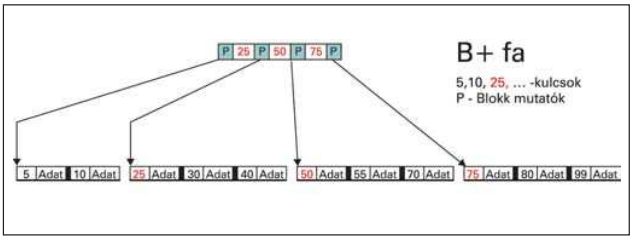
Az elágaztatási ütem (n) a csomópontok számát jelöli, amelyekre az egyes szintek csomópontjai mutatnak (*1. ábra*). Ha minden egyes csomópont az alatta lévő szinten `n` csomópontra mutathat, akkor a tetejéről elindulva a gyökércsomópont a következő szinten `n` számú belső csomópontra mutat, amelyek mindegyike `n` számú belső csomópontra mutat az azt követő szinten és így tovább. Az `m` szintű belső csomópont `nm` levélcsomópontra tartalmazhat hivatkozást, amelyek az utolsó szinten rendelkeznek elemekkel. Minél több adatot próbálsz tárolni egy fában, annál nagyobb mezőkre van szükséged a kulcsoknál, amikkel különbséget lehet tenni az elemek között, majd pedig az adott elem valamelyik része (egy bizonyos eltolásnál) kijelölhető. Ez azt jelenti, hogy a kulcsaidnak nagyobbaknak kell lenniük, ennek következtében nő az elágazási szám (kivéve, ha a kulcsokat tömöríted, de ezzel a következő változat megjelenéséig még várni kell).
Az 1. ábra első része egy négyszintű fát ábrázol, aminek az elágazási száma n = 1. Csupán négy darab csomóponttal bír, ezek a vörös gyökércsomóponttal indulnak, átlépve a sötétvörös, vagyis belső csomópontba, majd a kék gallycsomópontba, végül pedig a zöld levélcsomópontba, ami a tényleges adatot tartalmazza.
A második négyszintű fa (amelynek az elágazási száma n = 2) egy gyökércsomóponttal kezdődik, amit két belső csomópont követ, amelyek mindegyike két gallycsomópontra mutat (ez összesen négy gallycsomópontot jelent), és mindegyikük tovább egy levélcsomópontba, ami összesen nyolc levélcsomópontot jelent. Végül pedig egy négyszintű, n = 3 elágazási számú fát láthatunk, melynek egy gyökércsomópontja, három belső csomópontja, 9 gallycsomópontja, és 27 levélcsomópontja van.



1. ábra Három darab négyszintű kiegyensúlyozott fa



2. ábra Egy B-fa



3. ábra Egy B+fa

A B+ fák jobbak a B-fáknál

A belső csomópontokban nemcsak mutatókat és kulcsokat lehet tárolni, hanem a kulcsokhoz tartozó elemeket is. Erre volt képes az eredeti B-fa algoritmus (*2. ábra*). Ezt követően találták ki a B+ fákat, amelyek csak mutatókat és kulcsokat tárolnak a belső csomópontokban, a kapcsolódó elemek pedig levélszinten tárolódnak (*3. ábra*).

A B+ fák elágaztatási jobb a B-fákénál

Az elágaztatás növelhető, ha a belső csomópontokban csak mutatókat és kulcsokat helyezünk el, továbbá nem hígítunk az elemek adataival. A nagyobb elágaztatásnak köszönhetően a belső csomópontok hatékonyabban gyorstárazhatók, mivel így kevesebb belső csomópontot kell kezelni. Az emberek erre gyakran azt mondják, hogy „de a B-fák elemeket gyorstáraznak, és az effajta gyorstárazás éppoly hasznos". Általánosságban azonban nem ez a helyes válasz. Ha az általánosságot

vesszük alapul, a vita eldöntése még nehezebb lehet. De mielőtt erre jobban kitérnénk, tekintsük át a gyorstártervezés alapjait! Tételezzük fel a következőket:

- Kétcsoportnyi elemmel rendelkezel: A-val és B-vel.
- Tegyük fel, hogy a két csoportból változó rendszerességgel szükségesek az elemek: bizonyos dolgokra gyakrabban van szükség, másokra ritkábban, és az egyes csoportba tartozó elemek idővel változhatnak.
- Bizonyos elemeket kéznél tartunk, miután a korlátozott méretű gyorstárból előszedtük őket.
- Az A csoportban található elemekhez olykor egy B csoportbeli elemet kell kötni, ami annyit tesz, hogyha szükségünk van egy elemre az A csoportból, akkor annak a B csoportbeli társára is szükség lesz.

Így nő az A csoportból a gyakran használt elemek tárolásához szükséges gyorstár mérete. Ha erős kötődés van két szükséges elem között, amelyek a csoportok egyikében egymáshoz kötöttek, és ez a kötődés erősebb, mint a gyorstárazáshoz felhasznált erősforrásokon elért nyereség, ahol A és B az LRU (Least Recently used: újabban legkevésbé használt) algoritmusnak megfelelően gyorstárazódik, az hasznos lehet. Ha azonban nincs efféle kapcsolat, akkor kifejezetten rossz. Az LRU egy olyan algoritmus, amely az újabban kevésbé használt elemeket szórja ki a gyorstárból, így szabadít fel helyet. Az LRU különféle változatai az operációs rendszerek fejlesztésekor használt leggyakoribb gyorstárazási algoritmusok közé tartoznak.
De várjunk csak! Mi történik akkor, ha a B csoportból is szükséged van valamilyen elemre, s ezért jó lenne, ha innen is néhány elem a gyorstárba kerülne, tehát a B csoport valamelyik szeletére lenne szükség? Ebben az esetben az a gond, hogy kapcsolat nélkül nem valószínű, hogy neked a B csoportból arra az elemre lesz szükséged, ami valamelyik A-beli elemhez kötődik. Ha kiválasztasz valamit a B-ből, akkor az a gyorstárban tárolódik, és ezáltal az LRU-ra épülő gyorstár hatékonysága csökken, kivéve, ha a gyorstárazásért egy másik, legalább olyan jó algoritmus felel, mint az LRU. Amikor azt próbáljuk kiválasztani, hogy az A csoportnak megfelelően a B csoportból melyik elem gyorstárazása lenne ajánlatos, gyakran megesik, hogy az adott algoritmus nem olyan jó, mint az LRU, így megint bajban vagyunk.
A dolgoknak kevésbé hatékony összekötése, amikre rendszertelenül van szükségünk, a számítástechnikai világon kívül gyakori. Például tegyük fel, hogy szereted a pattogatott kukoricát és a szusit, de csak bizonyos napokon eszed őket és teljesen véletlenszerűen. Tételezzük fel, hogy a megnézendő filmeket is teljesen véletlenszerűen választod ki. Vegyük úgy, hogy a mozi megköveteli, hogy az adott, véletlenszerűen kiválasztott film esetében csakis pattogatott kukoricát ehetsz, és nem ehetsz szusit, amit pedig a sarki étteremben megkapnál. Ez lenne a társadalmilag legelőnyösebb rendszer? Tegyük fel, hogy a hot dogok minősége az egyes hotdog-árusok között véletlenszerűen változik. Ha a legjobb moziban egy bizonyos este csak a moziban készült hot dogot eheted, és a külső árusnál vásárolt hot dogot nem viheted be a mozi területére, az szerinted a társadalmilag legelőnyösebb? Előnyös-e ez számodra? A szorosan egymáshoz kapcsolódó dolgok összerendelése néha a teljesítmény szempontjából is előnyös. Számos fájlrendszer összerendeli a fájlméretadatokat a fájlnévadatokkal, ami úgy tűnik, hogy jól működik, legalábbis jobban, mint ahogyan az LRU működne.
Gyakori hiba a gyorstárak tervezésekor, hogy olyan dolgokat kötünk egymáshoz, amelyek egyébként nem állnak kapcsolatban, de ez még nem elég ahhoz, hogy megértsük, miért jobbak a B+ fák a többinél. Belső csomópontok esetében egyetlen csomópontban több mutatót is tárolunk, ezáltal a mutatók nem külön-külön kerülnek a gyorstárba. Vitatható, hogy a mutatók és az általuk hivatkozott elemek szorosabban összefüggenek-e, mint több különböző mutató. Remélem, a leírtak valóban tanulságosnak bizonyulnak, azonban még mindig meg kell értenünk egy gyorstárazási alapelvet.

Saját meghatározásom a gyorstár hőmérsékletére

Legyen a gyorstár hőmérséklete egy olyan érték, ami azt határozza meg, hogy milyen gyakran férünk hozzá az adott területhez, megszorozva a lemezről való beolvasás költségével, végül elosztva a felhasznált bájtok számával. Talán észrevettél egy szándékos pontatlanságot az iménti meghatározásban: hogyan lehetnek a kisebb elemek forróbbak beolvasásuk teljesítményigényességének következtében? A gyorstár-hozzáférés hőmérsékletére egyéb meghatározások is elképzelhetők, jelen cikk esetében azonban ez tűnik a legmegfelelőbbnek.
Ha két különböző típusú dolgot tárolunk egy csomópontban, amelyeknek különbözik az átlaghőmérséklete, akkor két külön csomópontba helyezésükkel a gyorstárazás hatékonyabbá tehető. Tegyük fel, hogy R bájtnyi RAM áll rendelkezésre gyorstárazásra, míg D bájtnyi lemezterületünk van. Tegyük fel, hogy a kérések nyolcvan százaléka a legutoljára használt dolgokra hivatkozik, ezek a csomópont H (vagyis forró) bájtjai. A nagyobb teljesítmény érdekében a H-t az R-hez képest csökkentjük. Ha a gyakran használt adatokat egyenletesen szétszórod, akkor nagyobb méretű gyorstárra lesz szükséged, és a gyorstár is veszít a hatékonyságából.

Gyorstárazási alapelvek

Ha növeljük a csomópontok közötti hőmérséklet-eloszlást, akkor kisméretű gyorstárból is nagyobb teljesítmény tudunk kihozni. Ha kétfajta dolognak eltér a hőmérsékletátlaga, akkor külön csomópontokban helyezzük el őket, így növelve a rendszer egészében a hőmérséklet-eloszlást.
Ha minden egyéb egyenlő, és két különböző típusú dolgot tárol több példányban egyetlen csomópontban, akkor a gyorstár hatékonyabbá tehető, ha több különböző csomópontban osztjuk el őket.

Mutatók csomópontokra

A csomópontokra hivatkozó mutatók gyakran hívódnak meg az általuk elfoglalt bájtok számához hasonló gyakorisággal. Vegyük azt az esetet, amikor mindhárom átjáró esetében mutatókat kell használni, amelyek az alsóbb szinten lévő csomópontokat érik el, és kisebbek, mint azok a csomópontok, amikre hivatkoznak. Ha csak csomópontmutatókat helyezünk el, és a kulcsokat a belső csomópontokra korlátozzuk, a csomópontok sűrűsége nő. Mivel a mutatókra méretükhöz képest négyzetesen több alkalommal van szükség, mint a fájlok tartalmát tároló többi elemre, a mutatók és a cikkelyek adatai között jelentős a hőmérsékletátlaguk közötti különbség.
A korábban tárgyalt gyorstárazási alapelveknek megfelelően a kétfajta hőmérséklet-átlagú csoport (a mutatók és az elemek adatai) elágaztatása növeli a gyorstár hatékonyságát.
Most talán azt kérdeznéd, hogy miért nem a valódi hőmérsékletadatok alapján végezzük az elágaztatást a típus szerinti helyett? Azért, mert a típus csak a hőmérséklettel függ össze? Azt tesszük, amit a legegyszerűbb kódolni, nem csak a hőmérséklet-eloszlást véve alapul.

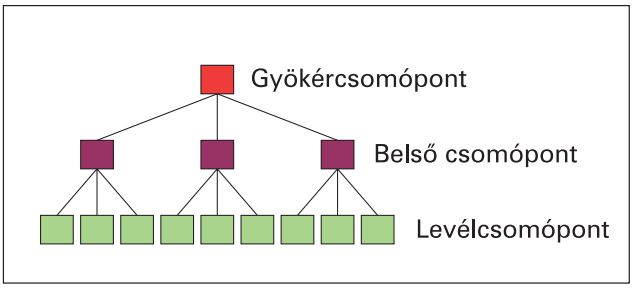
Néhány fatervezet úgy rendezi át a fát, hogy a magasabb hőmérsékletű elemek magasabban vannak, mint az alacsonyabb hőmérsékletű mutatók. Az elemadatok és a csomópontmutatók közötti hőmérséklet-különbség olyan mértékű, hogy az effajta tervezeteket én nem tartom versenyképesnek, emellett a megvalósításuk is bonyolultabb. A négyzetes nagyságú átlaghőmérséklet-különbséget véve alapul azt gyanítom, hogyha tévedek is, ez mégsem elegendő ahhoz, hogy foglalkozzunk vele. A többi fatervezet mintegy oldaljegyzetként megemlíti, hogy a vándorló elemek hőmérséklet szerint magasabban helyezkednek el a fában. Ha valaki pusztán csak a hőmérséklet szerint válogatna, de a szintek változtatása nélkül, az talán tényleg hatékonyabb lenne. Amennyiben valaki nem rendelkezne versenyképes tervezési alapokkal az elemek egymás körüli csoportosítására (ami néhány alkalmazás esetében igaz), és ha valaki az elemeket ahelyett, hogy egyenként férne hozzájuk, csomópontonként próbálja elérni, akkor érdemes lenne egy csomópont-újracsomagolót birtokolnia, hogy az elemadatokat hőmérséklet szerint csomópontokba rendezze.

A B+ fák jobban végzik a gyorstárazást, mint a B-fák

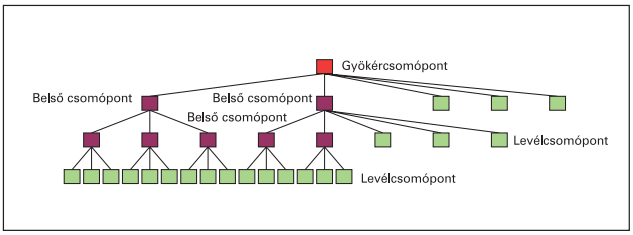
A B+ fák külön csomópontokban osztják el a mutatókat és az adatokat. A fában csomópontokra hivatkozó mutatók általában forróbbak, mint a fában található egyéb dolgok (mintegy négyzetes arányban). Mindazonáltal a korábban tárgyalt gyorstárazási alapoknak megfelelően a mutatók és az adatok szétválasztása hatékonyabb gyorstárazást eredményez. A számítógépiparban a B+ fákat a gyakorlatban jobbnak ismerik el, mint a B-fákat, mint ahogyan ezt a felvázolt elmélet is megjósolja. Az is elfogadott bölcsességnek minősül, hogy a kiegyensúlyozott fák hatékonyabbak a kiegyensúlyozatlan fáknál. Egyelőre azonban még nem elfogadott, de a fentiek alapján mégis megjósolható, a BLOB-ok – ahogyan az adatbázisipar hívja őket – rossz hatással vannak a teljesítményre. Erről és arról, hogy mik azok a BLOB-ok, hamarosan bővebben is szót ejtek.

Mit takar a kiegyensúlyozott?

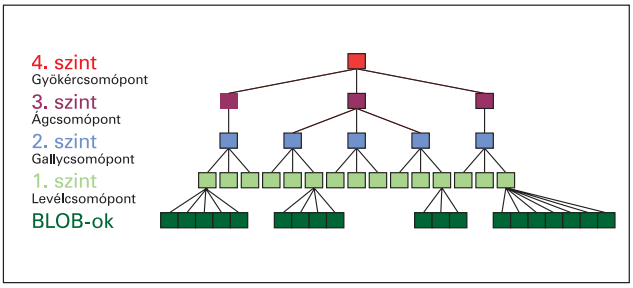
Ezt a fogalmat többféle, egymással kapcsolatban nem álló dologra is alkalmazzák, mint ahogyan a kiegyensúlyozott fák szakirodalma is megemlíti. E dolgok két legismertebbike a kiegyensúlyozott magasságok és a kiegyensúlyozott területek, amelyek a fa csomópontjaiban találhatók. Sajnálatos módon ezek a különféle meghatározások sok félreértést okoznak a szakirodalom olvasóinak – ezt én megpróbálom elkerülni.
A magasság-kiegyensúlyozott fák azok, amelyek esetében a gyökértől elindulva a levélcsomóponthoz vezető minden útvonal azonos hosszúságú. A hosszúság alatt a gyökércsomóponttól a levélcsomópontig bejárandó csomópontok számát értjük. Például az első ábrán a fa magassága 4, míg a *4. ábrán* a fa magassága 3, míg az egy csomópontból álló fa magassága 1. A legtöbb algoritmus a magasságmérés elvégzése céljából a fát csak a tetején növeli, ezáltal a fa sosem esik ki a magassági egyensúlyból.
Az *5. ábrán* egy kiegyensúlyozatlan fa látható. A fa kezdetben talán kiegyensúlyozott volt, de a különböző törlések következtében elvesztette néhány belső csomópontját. Illetve az is elképzelhető, hogy azért vált kiegyensúlyozatlanná, mivel anélkül lettek hozzá új elemek adva, hogy ügyeltek volna a kiegyensúlyozottságra.
A hagyományos adatbázismódszerekkel, ha egy csomópontnál nagyobb elemet szeretnénk tárolni, akkor ezt BLOB-okban tehetjük meg, ami azt eredményezi, hogy a fa kiegyensúlyozatlan lesz. A BLOB-ok jelentik az általános módszert arra, hogy



4. ábra Egy háromszintű fa

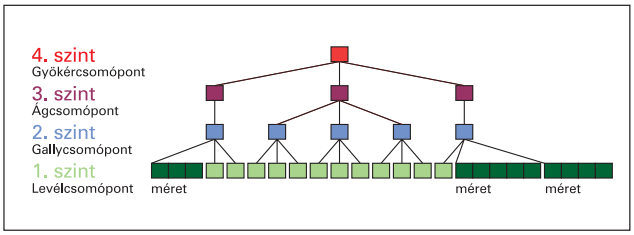


5. ábra Egy kiegyensúlyozatlan fa



6. ábra Négyszintű fa egy BLOB beszúrása után

nagyméretű elemeket tároljunk; működésük lényege pedig az, hogy a csomópont mutatókat tartalmaz azokra a csomópontokra, amelyek az elemet tartalmazzák. Az elemet tartalmazó csomópontokat levélcsomópontoknak hívják, ez minden "B*" fára jellemző.
A *6. ábrán* egy BLOB-ot szúrtunk be egy négyszintű fa levélcsomópontjába, ami annyit tesz, hogy a levélcsomópontban mutatókat helyeztünk el, amelyek a fájladatokat tartalmazó részekre hivatkoznak. A Reiser fájlrendszer 3-as változatának a fái így működtek.
Ez elég jelentős változás, amit mégis a teljes adatbázis-közösség elfogadott. Az itt leírt gyorstárazási alapelveknek megfelelően így csökkenthető a mutatók és elemadatok elágaztatása, de ez ront a gyorstár hatékonyságán. Emlékezzünk rá: a gyorstárazás alapelvei szerint ez rossz elgondolás. Tehát mindazon okokból kifolyólag, amelyek szerint a B+ fák jobbak a B-fáknál, a Reiser fájlrendszer 4-es változatának fái is jobbak a 3-as változat fáinál, bár nem olyan nagy mértékben.
Hogy ezt jobban megértsük, a *7. ábrán* egy Reiser4-es fa látható 3-as elágazási számmal, egy BLOB-bal az első szintű levélcsomópontban, valamint egy erre hivatkozó mutatóval a hármas szintű gallycsomópontban. Ebben az esetben a BLOB mezői folyamatosan helyezkednek el. A tárterület érdekében ez a többi levélcsomópont alatt helyezkedik el, de a hozzá tartozó mutató egy második szintű gallycsomópontban található, mint minden más elem mutatója is.
Bár elfogadott dolog, hogy a B+ fák jobbak a B-fáknál, az mégsem olyan elterjedt, hogy a BLOB-ok rosszul vannak megter-



7. ábra A Reiser4 a BLOB-okat az első szintű levélcsomópontokban tárolja

vezve, és az ezzel kapcsolatos tévhit jellemző az adatbázisiparra. Gray és Reuter azt mondja, hogy a külső memória keresésének az a jellegzetessége, hogy az egyes oldalak számát az általános (vagy leghosszabb) keresési úton csökkentsük, méghozzá olyan módon, hogy egy tetszőleges keresési útra csökkentjük az egyes oldalak számát, kisebbíti a valószínűségét annak, ha valamit közvetlenül a lemezről szükséges beolvasni (lásd a *Kapcsolódó címeket*).

Ezzel a magasság-kiegyensúlyozottság hatékonyságáról szóló elemzéssel az a bajom, hogy nem árulja el, hogy egy mérsékelten kiegyensúlyozott fával is lehet boldogulni – feltéve, hogy nem növeled meg nagymértékben a belső csomópontok számát. A gyakorlatban a kiegyensúlyozatlan fák csakugyan több belső csomóponttal rendelkeznek. A legtöbb mérsékelten kiegyensúlyozatlan fa esetében valamennyivel több memórián belüli lépdelésre van szükség a fában, ugyanakkor az I/O-terhelés nagymértékben nő, mivel több belső csomópontot kell feldolgozni, amiket – mivel így már túl sok csomópont van – nem lehet a gyorstárban tárolni.

De ha az összes BLOB-ot egyetlen helyen tárolnánk a fában, akkor ezáltal a csomópontok száma nem nőne jelentős mértékben, mivel az adatoknak minden más levélnél alacsonyabb szinten tartásából adódó teljesítménycsökkenés kis mértékű lenne. A fa bejárásának a költsége azonban – a RAM sebességétől függően – valamelyest nőne, de nem jelentős mértékben. A Reiser4-féle megközelítés nem az egyetlen lehetséges módja a belső csomópontszám csökkentésének. A BLOB-ok elágaztatása valószínűleg alapvetően megoldaná a gondot, ami azoknak a tervezőknek a hibájából keletkezik, akik a magasság-kiegyensúlyozott fák meghatározásának a lényegét figyelmen kívül hagyják. Valószínűleg az is nagy hatással lenne az I/O-teljesítményre, ha elkülönítenék azokat a BLOB-okat, amelyek nem állnak szoros kapcsolatban a fával. A Reiser4 esetében nem akartam arra korlátozni a fát, hogy az elemeket csak a méretük szerint ágaztassa el egymástól. De egy napon valaki majd biztos megpróbálja, és remélhetőleg elmondja nekünk, hogy mit tapasztalt.

A Reiser4 visszatér a magasság-kiegyensúlyozott fák klasszikus meghatározásához, vagyis minden egyes levélcsomóponthoz az út hossza egyenlő nagyságú. A Reiser4 nem színleli, hogy a nagyméretű elemeket tároló csomópontok nem a fa részei, annak ellenére sem, hogy a fa mutatókkal hivatkozik rájuk.

A Reiser4 a Reiser fájlrendszer 3-as változatához képest csökkenti a belső csomópontok és a mutatókat tartalmazó csomópontok számát. A 188 megabájtos Linux-rendszermag 2.4.1-es változatának forrásának tárolásához a ReiserFS v3 1629 csomópontot használt fel, ehhez a Reiser4-nek csak 164 csomópontra van szüksége. Ennek eredményeképpen a mutatók és csomópontok tárolásához szükséges RAM-mennyiség a Reiser4-nél döbbenetes mértékben mérséklődött.

Megjegyzés a cikk következő részeihez

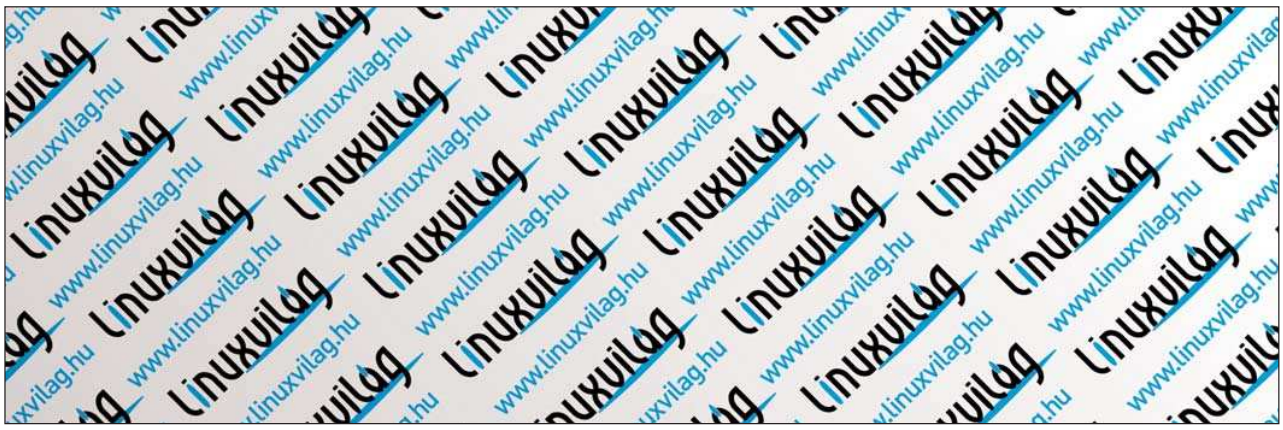
A soron következő cikkekben megtudjuk, hogy miért lényegesen jobb a Reiser4 teljesítménye, mint a ReiserFS v3-é, még teljesen üres gyorstár mellett is. Megtudjuk, hogy miért jobbak a táncoló fák, mint a helyigényes kiegyensúlyozott fák, és azt is, hogy miként kezeljük a tranzakciókat olyan módon, hogy ugyanakkor nagymértékben csökkentjük a kétszer leírandó adatok mennyiségét.

Linux Journal 2003. május, 109. szám

Hans Reiser (reiser@namesys.com)
1979-ben csatlakozott az UC Berkeley-hez, és „Rendszerezésre" szakosodott, ami egy egyéni ágazat. Az elméleti modellek fejlődését tanulmányozza.

KAPCSOLÓDÓ CÍMEK

Jim Gray és Andreas Reuter Transaction Processing: Concepts and Techniques. Morgan Kaufmann Publishers, Inc., 1993. – Ez egy régi, de jó könyv a tranzakciókról. Elérhető a ➲ http://www.mkp.com/books_catalog/catalog.asp?ISBN=1-55860-190-2 címen.



Processzorhoz kötés

Bizonyos folyamatok egyetlen processzorhoz kötése egy új rendszerhívás segítségével.

A Linuxnak az a képessége, hogy a segítségével folyamatokat köthetünk egyetlen processzorhoz, már nagyon régóta várt tulajdonság volt. Ezt a képességet processzorhoz kötésnek nevezzük, ami azt jelenti: „ez a folyamat csak ezen a bizonyos processzoron futhat”, vagy pedig „ezek a folyamatok csak ezeken a processzorokon futhatnak, de a nullás processzoron nem”. Az ütemező ennek megfelelően betartja a kikötéseket, és a folyamatok csak a számukra kijelölt processzorokon fognak futni.
Más operációs rendszerek, mint például a Windows NT, már régóta biztosítanak processzorhoz kötéssel kapcsolatos rendszerhívásokat. Ennek következtében a Linuxban is egyre inkább kívánatossá vált, hogy ilyen képességekkel bővítsék. Végül a 2.5-ös rendszermagban megjelentek a folyamatok processzorhoz kötését meghatározó és lekérdező rendszerhívások. A cikkből kiderül, hogy miért szükséges, hogy ismerjük a processzorhoz kötéssel kapcsolatos képességeket, és később azt is megtudhatjuk, hogyan használhatjuk ki ezeket az új képességeket. Ha nem vagy programozó, vagy csak akad egy olyan programod, amit nem vagy képes módosítani, bemutatok neked egy segédeszközt, amellyel egy adott PID-del rendelkező folyamatod processzorhoz való kötését állíthatod be. Végül pedig megvizsgáljuk a processzorhoz kötéssel kapcsolatos rendszerhívások megvalósítását.

Gyenge és erős processzorhoz kötés

Kétféle processzorhoz kötés létezik: a gyenge és az erős.
A gyenge, amit természetes kötésnek is hívunk, az ütemezőnek azt a tulajdonságát hivatott jellemezni, hogy egy folyamat esetében az ütemező mindig arra törekszik, hogy az adott folyamat a lehető leghosszabb ideig ugyanazon a processzoron fusson. De ez pusztán próbálkozás, mivel ha bármi közbejön, a folyamat rögtön költözik is a másik processzorra. A 2.5-ben található O(1) ütemező kiváló természetes kötési képességekkel bír, amelynek a 2.4-es rendszermag pontosan az ellenkezője, ahol is a processzorhoz kötés meglehetősen gyenge volt. Ha ez a természetes kötés gyenge, létrejön a pingponghatás. Ez annyit tesz, hogy az ütemező a folyamat minden egyes meghívásakor más-más processzoron futtatja a folyamatot. Az *1. táblázatban* egy rossz processzorkötésű ütemezés látható; a *2. táblázat* azt mutatja, hogyan néz ki a jó processzorkötésű ütemezés. Ezekkel ellentétben az erős processzorkötés az, amit a processzorkötéssel kapcsolatos rendszerhívások biztosítanak. Az erős kötés kötelesség, amit az ütemezőnek kötelező érvénnyel be kell tartania. Ha például egy folyamat a nullás processzorhoz van kötve, akkor minden esetben csakis azon futhat.

1. táblázat **A pingponghatás**

	time 1	time 2	time 3	time 4
Process A	CPU 0	CPU 1	CPU 0	CPU 1

2. táblázat **A jó processzorhoz kötés**

	time 1	time 2	time 3	time 4
Process A	CPU 0	CPU 0	CPU 0	CPU 0

Miért van szükség a processzorhoz kötésre?

Mielőtt részletesebben megismerkednénk az új rendszerhívásokkal, előtte tisztázzuk, hogy egyáltalán miért van szükség rájuk. Elsődleges előnyük az, hogy a gyorstárazás teljesítménye növelhető. Mint mondtam, az O(1) ütemező erősen törekszik arra, hogy a folyamatok egy processzoron maradjanak, és lehetőség szerint ott is tartja őket. De bizonyos teljesítményigényes helyzetekben – mint amilyen mondjuk egy nagy adatbázis-kiszolgáló futtatása, vagy egy temérdek szállal dolgozó Java-kiszolgáló – nagyon fontos lehet, hogy ezt a processzorhoz való kötést a legszigorúbban kezeljük. A több processzoros számítógépek keményen megdolgoznak azért, hogy a processzorok gyorstára érvényben maradjon. Az adatok egyszerre csak az egyik processzor gyorstárában találhatók meg, máskülönben a gyorstár elveszíti összehangoltságát, és ez bizonytalansághoz vezet azt illetően, hogy melyik processzor gyorstára tartalmazza a rendszermemóriának leginkább megfelelő másolatot. Ennek következtében, ha az egyik processzor egy sornyi adatot helyez el a gyorstárában, akkor az összes többi processzornak a helyi gyorstárában érvénytelenítenie kell azt a bizonyos sort. Ez az érvénytelenítés meglehetősen kellemetlen és költséges. De az igazi gond akkor jelentkezik, ha egy folyamat ugrál a processzorok között, ami érvénytelenítések sorozatával jár, és a szükséges adat sosem található meg a gyorstárban. Így a gyorstár hibázási gyakorisága nagyon nagyra nő. A processzorhoz kötés ettől véd meg, és ezáltal növeli a teljesítményt.
A processzorhoz kötés másik nagy előnye első pillantásra a fentiek egyenes következményének tűnik. Ha több szál fér hozzá egyszerre ugyanahhoz az adathoz, akkor előnyös lehet, ha a szálak egy processzoron futnak. Így biztosított, hogy a szálak nem érvénytelenítik a különböző processzoron lévő adatokat, és nem okoznak hibákat a gyorstárban. Ez ugyanakkor SMP-rendszerek esetén csökkenti a többszálúságból következő teljesítménynövekedést. A gyorstárazásból nyert teljesítménynövekedés talán megéri azt a veszteséget, amit a szálak egymás utáni végrehajtása következtében el kell szenvednünk. A harmadik és egyben utolsó előny valós idejű, illetve időérzékeny alkalmazások esetében lehet számottevő. Ebben a megközelítésben minden egyes rendszerfolyamat a rendszerben bizonyos processzorokhoz van kötve. Egy bizonyos alkalmazás pedig a maradék processzorokhoz. Általános esetben egy kétprocesszoros rendszerben ez a bizonyos alkalmazás egy processzorhoz van kötve, míg minden más folyamat a másik processzorhoz. Így biztosított, hogy erre a bizonyos alkalmazásra összpontosul az adott processzor összes figyelme.

Az új rendszerhívások elérése

A rendszerhívások meglehetősen újak, ezért egyelőre nem minden rendszeren érhetők el. Legalább 2.5.8-pre3-as változatszámú rendszermag és 2.3.1-es glibc szükséges (bár a 2.3.0-s glibc támogatja a rendszerhívásokat, egy hibát is magában rejt). 2.4-es rendszereken az új rendszerhívások még nem érhetők el, de egy folt már rendelkezésre áll, ami a ➲ http://www.kernel.org/pub/linux/kernel/people/rml/cpu-affinitycímről tölthető le, illetve megtalálható az 51. CD Magazin/Processzor könytárában.
A frissebb terjesztések közül több is magában foglalja ezeket a szolgáltatásokat. A Red Hat 9-ben például már a rendszermag és a glibc is tartalmazza a rendszerhívásokhoz szükséges támogatást. A valós idejű megoldások közül például a MontaVista Linux támogatja az új felületet.

Kötődési maszkok

A legtöbb rendszeren – beleértve a Linuxot is – a processzorhoz kötést bitmaszkok segítségével lehet beállítani. A bitmaszk *n* számú bitből áll, ahol az egyes bitek be-, illetve kikapcsolt állapotától függnek bizonyos szolgáltatások. Például a processzorhoz kötést (32 bites számítógépeken) egy 32 bites bitmaszk határozza meg. Az egyes bitek azt jelölik, hogy az adott folyamat kötve van-e egy bizonyos processzorhoz vagy processzorokhoz. A biteket jobbról balra kell számolni, a 0-s bittől a 31-es bitig, és ennek megfelelően a nullás processzortól a 31. processzorig. Például:

```
11111111111111111111111111111111 = 4 294 967 295
```

Ez az alapértelmezett processzorhoz kötési maszk minden folyamat esetében. Mivel minden bit be van kapcsolva, a folyamat bármelyik processzoron futhat.
Ugyanakkor, ha a bitmaszk

```
00000000000000000000000000000001 = 1
```

akkor sokkal szigorúbb megkötések érvényesek. Minek utána csak a 0. bit van bekapcsolva, a folyamat csak a nullás processzoron futhat, vagyis a kötési maszk a folyamatot a nullás processzorhoz köti. Remélem, érthető.
Mit jelent a következő két maszk tízes számrendszerben értelmezve? Mi történik, ha ezt a két maszkot egy folyamat kötési maszkjaként használjuk fel?

```
10000000000000000000000000000000
00000000000000000000000000000011
```

Az első maszk 2 147 483 648-cal egyezik meg, mivel a 31. bit van bekapcsolva, és a folyamatot a 31. processzorhoz köti.
A második bitmaszk értéke 3, és a folyamatot a nullás és az első processzorhoz köti.
A Linux processzorkötési felülete a fentiekhez hasonló bitmaszkot alkalmaz. A C nyelv azonban sajnos nem támogatja a bináris állandókat, így mindig a decimális vagy hexadecimális, azaz tízes vagy tizenhatos számrendszerbeli állandókat kell használni. A fordító talán figyelmeztetni fog, ha olyan nagy állandót próbálsz használni, amelyik a 31. processzorhoz köti a folyamatot, de ne aggódj, működni fog így is.

Az új rendszerhívások használata

Az új rendszermag és glibc esetén az új rendszerhívások használata meglehetősen egyszerű:

```
#include <sched.h>

long
sched_setaffinity(pid_t pid, unsigned int len,
                  unsigned long *user_mask_ptr);

long
sched_getaffinity(pid_t pid, unsigned int len,
                  unsigned long *user_mask_ptr);
```

Az első rendszerhívást arra használjuk, hogy egy folyamat kötéseit beállítsuk, míg a második a már beállított kötéseket kérdezi le.
Mindegyik rendszerhívás estében a `PID` kapcsoló annak a folyamatnak a PID-jét jelenti, amelynek a tulajdonságait le akarod kérdezni, vagy változtatni akarsz rajtuk. Ha a PID nulla, akkor a hívó folyamatra vonatkoznak a beállítások.
A második kapcsoló a processzorkötési maszkban található bitek számát jelöli, amely jelenleg 4 bájt (32 bit). Ez a kapcsoló azért szükséges, mert ha a jövőben a rendszermagban változni fog az erre a célra fenntartott bitek száma, akkor a már elkészült programok az új környezetben is helyesen fognak működni. Nem lenne jó, ha meglévő programjaink egyszer csak működésképtelenné lennének. A harmadik kapcsoló egy hivatkozást tartalmaz, ami magára a bitmaszkra mutat.
Nézzük csak meg, hogyan kérdezhetjük le egy folyamat processzorhoz kötési bitmaszkját:

```
unsigned long mask;
unsigned int len = sizeof(mask);

if (sched_getaffinity(0, len, &mask) < 0) {
    perror("sched_getaffinity");
    return -1;
    }

printf("a processzorkötési maszkom:
       ➥%08lx\n", mask);
```

Afféle kényelmi szolgáltatásként a visszaadott bitmaszk `AND`-olva van a rendszerben található működőképes processzorok bitmaszkjával, így biztosan csak a működőképes processzorokhoz tartozó biteket látjuk bekapcsolva. Például egy egyprocesszoros rendszer a fenti hívásra mindig 1-gyel tér vissza (a 0. bit kivételével minden bit nulla).
A maszk beállítása éppen ilyen egyszerű:

```
unsigned long mask = 7;
➥/* processors 0, 1, and 2 */
unsigned int len = sizeof(mask);

if (sched_setaffinity(0, len, &mask) < 0) {
    perror("sched_setaffinity");
}
```

A példában a folyamatot a rendszerben lévő első három processzorhoz kötjük. Ezt követően meghívhatod a `sched_getaffinity()` függvényt, hogy megbizonyosodj róla, a fenti példa valóban működött-e. Mivel tér vissza a `sched_getaffinity()`, ha valójában csak két processzorral rendelkezel? Mivel tér vissza, ha csak egy processzorral? A rendszerhívás sikertelen, ha a bitmaszkban megjelölt processzorok legalább egyike nem létezik. A nullás maszk

1. lista A bind forrása

```
/* bind - egyszerű parancssoros eszköz
 * az alkalmazások processzorhoz kötésének
 * beállítására.
 */

#define _GNU_SOURCE

#include <stdlib.h>
#include <stdio.h>
#include <sched.h>

int main(int argc, char *argv[])
{
    unsigned long new_mask;
    unsigned long cur_mask;
    unsigned int len = sizeof(new_mask);
    pid_t pid;

    if (argc != 3) {
      fprintf(stderr,
              "usage: %s [pid] [cpu_mask]\n",
              argv[0]);
      return -1;
    }

    pid = atol(argv[1]);
    sscanf(argv[2], "%08lx", &new_mask);

    if (sched_getaffinity(pid, len,
                          &cur_mask) < 0) {
      perror("sched_getaffinity");
      return -1;
    }

    printf("pid %d's old affinity: %08lx\n",
           pid, cur_mask);

    if (sched_setaffinity(pid, len,
        ➥&new_mask)) {
      perror("sched_setaffinity");
      return -1;
    }

    if (sched_getaffinity(pid, len,
                          &cur_mask) < 0) {
      perror("sched_getaffinity");
      return -1;
    }

    printf(" pid %d's new affinity:
             ➥%08lx\n",
           pid, cur_mask);

    return 0;
}
```

mindig sikertelen hívást eredményez. Csakúgy, ha a 7. processzorhoz szeretnél kötni egy folyamatot, és nincs hét processzorod, a rendszerhívás ugyancsak sikertelen lesz.

A rendszerben minden folyamat processzorkötési maszkja lekérdezhető, azonban a maszk csak azokban a folyamatokban változtatható meg, amelyeknek te vagy a tulajdonosa, illetve bármely folyamat bitmaszkját megváltoztathatod, ha rendszergazdai jogosultságokkal rendelkezel.

Egy eszközt akarok!

Ha nem vagy programozó, illetőleg nem vagy képes módosítani a forrást, akkor is megváltoztathatod a folyamatok kötési tulajdonságait. Az első listában egy egyszerű parancssoros eszköz forráskódját találod, amellyel bármely folyamat kötési maszkját egyszerűen beállíthatjuk, ha tudjuk a folyamat PID-jét. Mint említettem, ehhez neked kell lenned az adott folyamat tulajdonosának, vagy pedig rendszergazdai jogosultságokat kell birtokolnod.
Az eszköz használata egyszerű, ha megtanulod a tízes számrendszerbeli kifejezésmódot:

```
bind pid maszk
```

Tegyük fel, hogy egy kétprocesszoros rendszerünk van, amelyben az 1600-as PID-del futó Quake-folyamatot a második processzorhoz szeretnénk kötni. Egyszerűen csak a következőt kell begépelnünk:

```
bind 1600 2
```

Legyünk igazán ravaszok

Az előző példában a Quake-et a rendszerünkben található egyik processzorhoz kötöttük. Hogy igazán gyors képet kapjunk, a rendszerünkben minden más folyamatot a másik processzorhoz kell kötnünk. Ezt megteheted kézzel is, vagy ha írsz egy trükkös parancsfájlt, de egyik sem igazán hatékony. Ehelyett használd ki, hogy a processzorhoz kötés a `fork()` során öröklődik, vagyis egy folyamat minden gyermeke ugyanahhoz a processzorhoz fog kötődni, mint a szülője.
Ennek a legtisztább módja, ha magát az `init`-et dolgozzuk meg egy kicsit, olyanformán, hogy a kívánt processzorkötési maszkot a rendszermag parancssorán keresztül adjuk át. Célunkat azonban egyszerűbben is elérhetjük, anélkül, hogy módosítanunk kellene vagy újra kellene fordítanunk az `init`-et. Helyette egyszerűen megszerkeszthetjük a rendszerindító parancsfájlt. A legtöbb rendszeren ez a parancsfájl a */etc/rc.d/rc.sysinit* vagy a */etc/rc.sysinit* fájlok egyike, amit az `init` először futtat le. Helyezzük el a `bind` példaprogramot a */bin* könyvtárban, és az *rc.sysinit* elejéhez adjuk hozzá a következő két sort:

```
/bin/bind 1 1
/bin/bind $$ 1
```

Ezek a sorok az `init`-et (aminek a PID-je 1) és az éppen futó folyamatot a nullás processzorhoz kötik. Minden jövőbeli folyamatnak ezek a folyamatok lesznek a szülői, így örökölni fogják a kötési maszkokat. Ezt követően a saját folyamatodat (legyen ez egy valós idejű nukleáris irányító rendszer vagy a Quake) a másik processzorhoz kötheted. Ezt követően minden folyamat a nullás processzoron fog futni, kivéve a saját folyamatunkat és annak gyermekeit, amelyek az egyes processzoron futnak. Ez azt jelenti, hogy így a teljes processzor csak a mi saját folyamatunkat fogja kiszolgálni.

Rendszermagbeli processzorkötés megvalósítása

Jóval azelőtt, hogy Linus felvette volna a processzorhoz kötési rendszerhívásokat, a rendszermag már támogatta és figyelembe vette a processzorhoz kötési maszkokat. Nem létezett azonban semmilyen felület, amin keresztül ezeket a maszkokat a felhasználó módosíthatta volna. Minden folyamathoz tartozó maszk a `task_struct` szerkezetben tárolódik `unsigned long`-ként, `cpus_allowed` néven. A `task_struct` szerkezetet a folyamat leírójának nevezzük. Ez tárol minden adatot az egyes folyamatokkal kapcsolatban. A processzorkötési felület egyszerűen csak kiolvassa és módosítja a `cpus_allowed` változót.

Bármikor, ha a rendszermag egy folyamatot át akar költöztetni egy másik processzorra, először ellenőrzi, hogy az adott folyamat futhat-e azon a másik processzoron. Ha a másik processzorhoz tartozó bit nincs bekapcsolva, akkor a processzor a folyamatot nem helyezi át. Ezen túlmenően, ha a processzorhoz kötési maszk megváltozik, és a folyamat éppen egy számára nem engedélyezett processzoron fut, akkor a folyamat soron kívül áthelyeződik egy számára engedélyezett processzorra. Ez biztosítja, hogy egy folyamat csak a számára engedélyezett processzoron kezdheti meg a futását, illetve csak olyan processzorra költözhet át, amelyen nincs tiltva a futása. Természetesen, ha csak egy processzorhoz van kötve a folyamat, akkor nem költözik sehova.

Összegzés

A 2.5-ösben megjelent processzorhoz kötési felület – és egyéb helyekre átültetett változatai – egyszerű, de mégis hasznos eszközt kínálnak arra, hogy meghatározhassuk, hogy az egyes folyamatok mely processzoron fussanak. A többprocesszoros géppel rendelkező felhasználók biztosan örülni fognak, hogy újabb lehetőség teremtődött arra, hogy rendszerüket még hatékonyabbá varázsolhassák, vagy arra, hogy a legfontosabb valós idejű alkalmazások mindig kapjanak processzoridőt. De az egyprocesszoros számítógéppel rendelkező felhasználóknak sem kell magukat mellőzve érezniük, ők is használhatják az új rendszerhívásokat, de ezáltal sem tesznek szert túl sok haszonra.

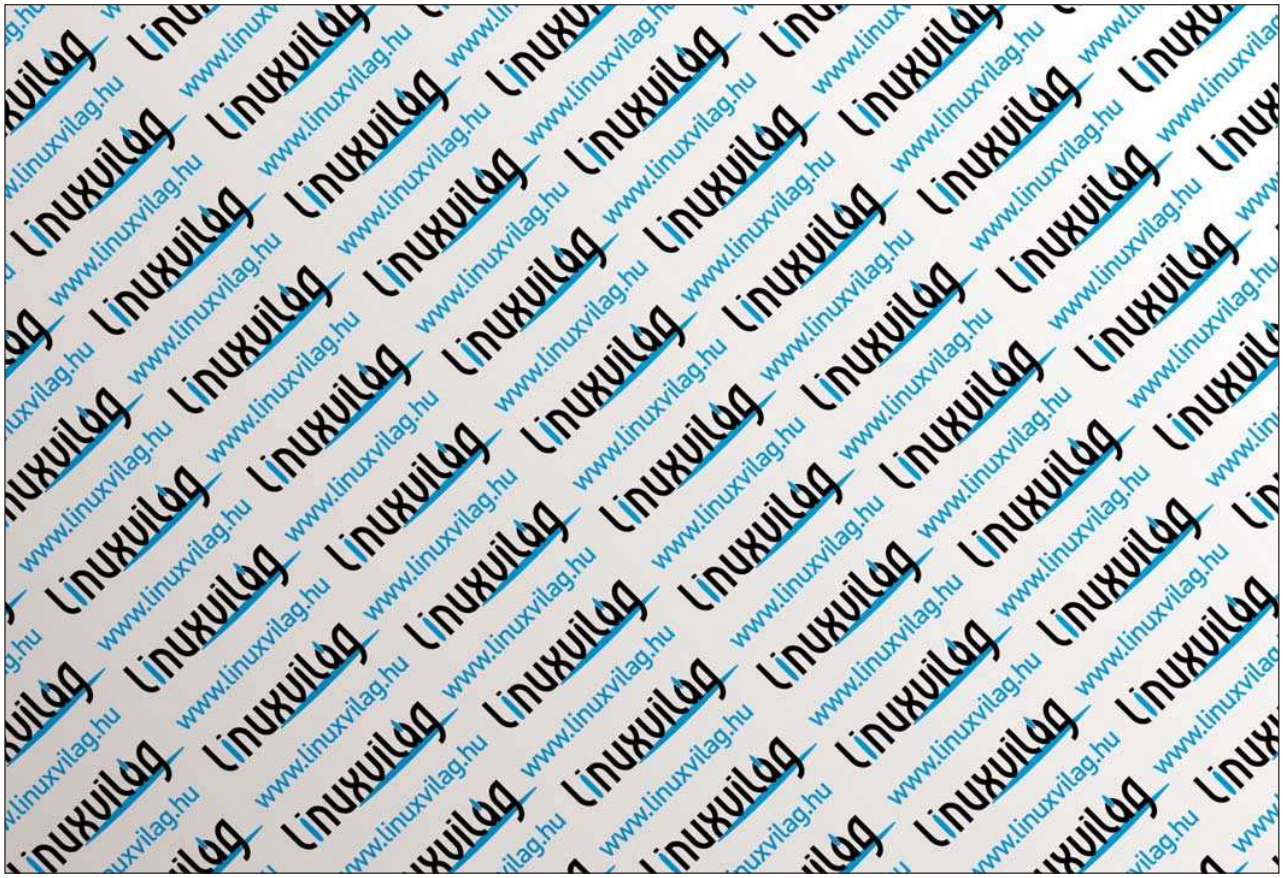
Linux Journal 2003. július, 111. szám

Robert Love
A Florida Egyetemen folytat matematikai és számítástudományi tanulmányokat, egyúttal a MontaVista Software-nél dolgozik rendszermag-fejlesztőként. Szeret fényképezni.

KAPCSOLÓDÓ CÍMEK

A Schedutils ütemezővel kapcsolatos eszköztár: ➲ http://tech9.net/rml/schedutils. A csomagban megtalálható a `taskset` nevű program, amellyel hatékonyan állítgathatjuk, a folyamatok processzorhoz kötését.

Processzorhoz kötési foltok a rendszermaghoz és a glibc-hez: ➲ http://www.kernel.org/pub/linux/kernel/people/rml/cpu-affinity



Az LDAP és a biztonság (1. rész)

Az OpenLDAP segítségével az összes alkalmazás kényelmes közös címtárat használhat. Helyes beállításával a hálózat biztonsága nemhogy csökken, hanem nőni fog.

Tegyük fel, hogy IMAP-kiszolgálót üzemeltetünk rengeteg felhasználóval, de nem szeretnénk minden felhasználónak teljes értékű fiókot létrehozni a kiszolgálón. Inkább valamiféle központi felhasználóhitelesítő szolgáltatást kellene használni, ami más célokra is jó lehet. Ha már itt tartunk, szükség lesz egy állandó elérésű (online) címjegyzékre szervezetünk levelező- és csoportmunkaprogramjai számára. Ezenkívül tegyük fel, a felhasználóknak titkosítóeszközökre is szükségük van, amelyek X.509-es tanúsítványokat használnak, valamint az egész szervezet számára kezelik a digitális tanúsítványokat. Elképzelhető, hogy létezik egyetlen szolgáltatás, ami mind a négy igényt ki tudja elégíteni? Az LDAP képes erre, sőt többre is. A nyílt forrás közössége abban a szerencsés helyzetben van, hogy rendelkezésére áll egy szabad, megbízható és teljes értékű LDAP-kiszolgáló és -ügyfél, ami a legtöbb Linux-terjesztésnek része: ez az OpenLDAP.

Az egyetlen hátrány, hogy az OpenLDAP egy bonyolult szörny. Értelmes használatához még több rövidítést és elvont elképzelést kell megismernünk a szokásos Unix-trükkökön kívül. Felszerelkezve a következő néhány hónapban megjelenő folytatással és egy kis elhatározással, nagyhatalmú LDAP-istenek lehetünk, akik egyszerre több vasat tartanak a tűzben, és a hálózatunk egyszerre lesz biztonságosabb és könnyebben használható. Tapasztalataim szerint a „biztonságosabb” és az „egyszerűbben használható” rendszer ritkán valósítható meg egyszerre, ezért is lelkesedem azért, hogy ebben a rovatban végre bemutathatom az OpenLDAP-t.

Az LDAP alapjai

Dióhéjban összegezve: az LDAP címtárszolgáltatást nyújt, azaz egy központi adatbázisban érhetők el az emberek, csoportok és a szervezetet alkotó más elemek adatai. Mivel minden szervezet különböző lehet, és nem feltétlenül ugyanazokat az adatokat tartják lényegesnek, a címtárszolgáltatásnak rugalmasnak és testreszabhatónak kell lennie. A megoldás ezért a feladat jellegéből adódóan bonyolult.

A címtárszolgáltatásra létezik egy protokoll, az X.500. Az X.500-at nagy és összetett szervezetek nagy léptékű címtárszolgáltatásaira tervezték. Ennek megfelelően az X.500 önmagában annyira nagy és bonyolult protokoll, hogy pehelysúlyú változata, az LDAP, amit az RFC 1777 ír le – és gyakorlatilag az X.500 protokoll részhalmaza –, sokkal inkább elterjedt, mint az eredeti X.500.

Az X.500 és az LDAP nyílt protokollok, miként a TPC/IP; egyik sem önálló termék. A protokollt valamilyen programnak kell megvalósítani, ez lehet magmodul, kiszolgálódémon vagy ügyfélprogram. A TCP/IP-hez hasonlóan nem minden LDAP-megvalósítás egyforma, még csak nem is feltétlenül képesek együttműködni egymással (módosítás nélkül). Most egy bizonyos LDAP-megvalósításról lesz szó, az OpenLDAP-ról, de tudnunk kell, hogy léteznek más programok is, amelyek szintén megvalósítják az LDAP-t. Például ilyen a Netscape Directory Server, a Sun ONE Directory Server és bizonyos korlátokkal a Microsoft Active Directory a Windows 2000 Serverben.

Szerencsére az LDAP-t bővíthetőre tervezték. Ha az egyik környezetben létrehozunk egy LDAP-adatbázist, az cseresza-batos lesz más LDAP-megvalósításokkal is, egyszerűen csak be kell állítanunk az adatbázis rekordformátumát vagy sémáját. Ez a téma a következő hónapban kerül sorra. Ennek köszönhetően gond nélkül futtathatjuk az OpenLDAP-kiszolgálót Linuxon, és a címjegyzéket elérhetővé tehetjük, mondjuk egy Netscape Communicatort futtató Mac-felhasználó számára is.

Az OpenLDAP beszerzése és telepítése

Az OpenLDAP hasznos és fontos eszköz, ennek következtében minden jelentős Linux-terjesztésnek része. Általában több csomagra bontják szét: az egyik a kiszolgálódémont, a másik az ügyfélprogramokat, a harmadik a fejlesztéshez szükséges programkönyvtárakat tartalmazza. Ez a cikk az LDAP-kiszolgáló felállítását tárgyalja, így magától értetődő, hogy terjesztésünk OpenLDAP-kiszolgáló csomagját telepíteni kell. Ezenkívül telepítsük az OpenLDAP futásidejű programkönyvtárait, ha azok nem részei a kiszolgáló csomagjának. Azt gondolhatnánk, hogy az OpenLDAP ügyfélprogramjainak telepítése kihagyható egy olyan kiszolgálón, ahol nincsenek helyi felhasználók, és az LDAP-műveletek a hálózaton keresztül fognak zajlani. Ez sajnos nem így van, kimondottan javasolt az ügyfélprogramokat is feltelepíteni, mert sokat segíthetnek a rendszer kipróbálásában és a hibakeresésben.

Red Hat alatt az OpenLDAP a következő csomagokból áll: openldap (OpenLDAP-programkönyvtárak, -beállítóállományok és a leírás); openldap-clients (OpenLDAP-ügyfélprogramok és -parancsok); openldap-servers (OpenLDAP-kiszolgálóprogramok); valamint az openldap-devel (fejlécállományok és programkönyvtárak fejlesztők számára). Bár a csomagok legtöbb függősége teljesen érthető (például glibc), két szükséges csomag talán még nincs telepítve: a cyrus-sasl és cyrus-sasl-md5, ezek az LDAP hitelesítő műveleteit hivatottak közvetíteni.

SuSE alatt a következő RPM-ekben helyezkedik el az OpenLDAP: openldap2-client (az ***n1*** könyvtárban a SuSE 7.3 és 8.0 esetén); openldap2 (ebben az OpenLDAP-programkönyvtárak és a kiszolgálódémonok csücsülnek, és az ***n2*** könyvtár részét képezik); openldap2-devel (a SuSE 7.3-ban az ***n2***-ben található, a SuSE 8.0-ban az ***n4***-ben érhető el). A Red Hatnél a már említett cyrus-sasl csomagot mindenképpen telepítsük fel, ami a ***sec1*** könyvtárban helyezkedik el.

A 7.3-as és 8.0-s terjesztésekben a SuSE az OpenLDAP 1.2-es változatát is közreadta a 2.0-s mellett. Mindenképpen a 2.0-s csomagokat telepítsük, hacsak nincs különleges okunk az OpenLDAP 1.2 futtatására. Ez a tanács nem vonatkozik a Red Hat- és a Debian-terjesztésekre, mert ezek kizárólag az OpenLDAP 2.0-t használják.

A Debian 3.0-s (Woody-) terjesztésben a megfelelő deb-csomagok a következők: libldap2 (OpenLDAP programkönyvtárak a Debian ***libs*** könyvtárából); `slapd` (az OpenLDAP-kiszolgáló a ***net*** könyvtárból); és az ldap-utils (OpenLDAP-ügyfélparancsok szintén a ***net*** könyvtárból). Ezenkívül még a ***libsasl7***

1. lista A /etc/openldap/slapd.conf testreszabott része

```
database        ldbm
suffix          "dc=wiremonkeys,dc=org"
rootdn          "cn=ldapguy,dc=wiremonkeys,
                 ➥dc=org"
rootpw          {SSHA}zRsCkoVvVDXObE3ewn19/
                 ➥Imf3yDoH9XC
directory       /var/lib/ldap
```

2. lista A slappasswd parancs

```
[root@mydirserver openldap]
# slappasswd -h {SSHA}
New password: *********
Re-enter new password: *********
{SSHA}16JhhIDajRc1cDwwa1t6o0ske8goj8Od
```

programkönyvtárra is szükség van a Debian ***libs*** könyvtárából. Amennyiben kedvenc terjesztésünk nem tartalmazza az OpenLDAP bináris csomagjait, vagy a legújabb OpenLDAP-ból szükségünk van egy adott tulajdonságra, amelyik nincs benne terjesztés OpenLDAP-csomagjában, vagy testreszabott OpenLDAP-csomagot szeretnénk, még mindig lefordíthatjuk az OpenLDAP hivatalos honlapjáról (➲ http://www.openldap.org) letölthető forrást.

A slapd beállítása és elindítása

Az OpenLDAP fő kiszolgálódémonjának a neve `slapd`, és az OpenLDAP telepítése után az első feladatunk ennek beállítása. A beállítások elsősorban a */etc/openldap/slapd.conf* állomány szerkesztésével végezhetők el.
A ➲ http://www.openldap.org/doc/admin20/guide.html címen megtalálható „OpenLDAP 2.0 Administrator's Guide” kitűnően elmagyarázza a `slapd` elindításának és futtatásának kezdőlépéseit a 2. fejezet 8. pontjától kezdve. A dokumentum a címtárszolgáltatásokat és az LDAP fogalmait írásunknál nagyobb mélységben tárgyalja.
Menjünk végig a lépéseken, hogy biztosan jól sikerüljön a kezdés. Az első teendő a ***slapd.conf*** szerkesztése – szemléltetésül tekintsük meg az *1. listát*. Láthatjuk, hogy a ***slapd.conf*** jellegzetes Linux-beállítóállomány: minden sora egy kapcsolónevet tartalmaz, amit egy érték követ. Az 1. listán szereplő első kapcsoló, a `database` azt adja meg, hogy milyen adatbázisháttérrel kívánjuk az OpenLDAP-t használni. Általában a legjobb az `ldbm`-et választani, ami a gyors `dbm` adatbázis-formátumot használja, de a `shell` (egyéni héjprogramhátterek) és a `passwd` (a */etc/passwd* használata háttérként) szintén érvényes beállítás.
Az 1. listán a következő kapcsoló a `suffix`, amely meghatározza, hogy milyen lekérdezések illenek erre az adatbázis-meghatározásra. Az itt megadott végződés a ***wiremonkeys.org***, amelyet az LDAP nyelvén balról jobbra értelmezett tartományelem- (`dc`) kifejezések sorozataként adunk meg. Más szavakkal, ha egy LDAP-ügyfél a `cn=bubba,dc=wiremonkeys,dc=org` megkülönböztető nevű (`dn`) gépet keresve lekérdezi a példakiszolgálónkat, akkor mivel a `dn` vége `dc=wiremonkeys,dc=org`, ez az adatbázis fog ráilleni. A megkülönböztető nevekről többet tudhatunk meg a „Gyorstalpaló az X.500-as nevezéktanról” című írásból (az 51. CD Magazin/LDAP könyvtárában található).
Az 1. lista következő két bejegyzése az LDAP-adatbázis felügyeletével kapcsolatos; a `rootdn` és a `rootpw` azt a felhasználónév és jelszópárost adja meg, amelyet a helyi vagy távoli parancsoknak kell megadniuk, ha felügyeleti műveletet akarnak végrehajtani az LDAP-adatbázison. Érdekes módon ez a bejegyzés csak erre a célra használatos. Nem jelenik meg a szabályos LDAP-adatbázis lekérdezésekben.
Ez felveti azt az ellentmondást, hogy akkor milyen módon lehet hitelesíteni azokat a műveleteket, amelyek a hitelesítési (LDAP) adatbázis feltöltéséhez szükségesek. Később, ha már feltöltöttük az LDAP-adatbázist valódi adatokkal, az egyik rekordot a ***slapd.conf*** hozzáférési listáit használva nevezzük ki felügyeleti fióknak, és töröljük a `rootdn` és `rootpw` bejegyzést. Ezt a lépést egy későbbi írásban tárgyalni fogjuk; pillanatnyilag a `rootdn` és a `rootpw` is megfelel.
Nagyon rossz ötlet a `rootpw` értékét egyszerű szövegben tárolni. Ehelyett a `slappasswd` parancsot kell használni, ami a jelszó kódolt formáját állítja elő, miként a *2. lista* is mutatja. Láthatjuk, hogy a `slappasswd` bekéri a jelszót, és a kódolt jelszót a `-h` kapcsolóval megadott algoritmus szerint előállítja és kiírja a képernyőre. A `-h` után kapcsos zárójelben kell megadni a kívánt algoritmus nevét. A lehetséges választék a slappasswd(8) súgóoldalon fel van sorolva. A `slappasswd` kimenetét közvetlenül bemásolhatjuk a ***slapd.conf*** állományba, én is pontosan ezt tettem az 1. lista `rootpw` értékének létrehozásakor.
Visszatérve az 1. listához, a következő érték a címtár meghatározásában a könyvtár. Nem meglepő módon ez azt adja meg, hogy a helyi állományrendszerben melyik könyvtárban legyen az LDAP-címtár létrehozva. Mivel a */var* a megszokott helye a növekvő állományoknak, mint a naplók vagy az adatbázisok, az 1. lista a */var/lib/ldap* értéket tartalmazza. Létező könyvtárat kell megadni, és az OpenLDAP felhasználójának és csoportjának – általában `ldep` és `ldap` – tulajdonában kell lennie. A jogosultságokat állítsuk a `0700 (-rwx------)` értékre.
Műszaki értelemben ennyi elég az elinduláshoz: az `ldap` indító-parancsfájljával próbáljuk meg elindítani a `slapd` démont. Ez leggyakrabban a */etc/init.d/ldap*, de terjesztésenként különböző lehet. Nyugodtan adjunk hozzá bejegyzéseket az LDAP-adatbázishoz az `ldapadd` parancs használatával – a korábban említett eljárás megmutatja, hogyan.
Mielőtt a hálózaton keresztül kezdenénk el kezelni és lekérdezni az LDAP-adatbázist, be kell állítani a TLS-titkosítást. Ez fontos, mert az OpenLDAP által használt egyszerű hitelesítési eljárás a hitelesítési adatokat titkosítás nélkül küldi át a hálózaton. Sajnos elfogyott a rendelkezésemre álló hely, ezért ez a téma a jövő hónapra marad. Aki nem tud addig várni, az olvassa el ***Vincent Danen*** Using OpenLDAP for Authentication írását a ➲ http://www.mandrakesecure.net/en/docs/ldap-auth.php címen, bár ez bizonyos tekintetben a Mandrake-et helyezi középpontba. Szintén tárgyalni fogom az LDAP-adatbázis szerkezetének meghatározásával kapcsolatos megfontolásokat, és az LDAP-adatbázis létrehozásának a módját.
Addig is sok szerencsét!

Linux Journal 2003. július, 111. 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.

Ismerjük meg a Monót!

A Mono haszna például abban áll, hogy segítségével a Linux képessé válik a Microsoft .NET rendszerével való munkára.

Amennyiben írtunk már valaha Linux-munkafelületen futtatható programot, vagy legalább megvizsgáltuk a lehetőségét, akkor ismert előttünk a nyelvi kötéseknek az a sokasága, ami a különböző grafikus felületek eszközkészletéhez rendelkezésre áll. A Linux grafikus felületére való programírásnak ez az egyik erőssége: nem vagyunk egy adott programozási nyelvhez hozzáláncolva. Sajnos hamarosan azt is észre kell vennünk, hogy a különböző nyelvi kötések az API igen eltérő készültségi fokával bírnak. Az egyik nyelven használt eszközkészlet egy másik nyelven esetleg még nincs támogatva – ez a „soknyelvűség” egyik hátulütője. Egy API karbantartásához szükséges munka mennyisége minden egyes nyelv esetén növekszik. Az eredeti API megváltoztatása vagy frissítése esetén a változást minden egyes nyelvi burkolón végig kell vezetni. Most pedig képzeljünk el egyetlen GUI-eszközkészletet, ami bármilyen programozási nyelvből anélkül elérhető, hogy API-burkolót kellene használnunk. Egy olyan eszközkészletről van szó, ami minden őt használó programozási nyelvnek ugyanazt a szolgáltatást nyújtja.

A Mono számos egyéb mellett ezzel a képességgel is rendelkezik, ezáltal teremtve meg felhasználója számára a nyelvfüggetlenséget és a nyelvek közötti párbeszéd lehetőségét.

A Mono története dióhéjban

A Mono története körülbelül két évvel ezelőtt kezdődött a Ximian nevű, Linux-programokra szakosodott cégnél. A Ximian leginkább a Ximian Desktopról, az Evolution naptár- és levelezőprogramjáról, a Red Carpet frissítőrendszerről és a vakbuzgó műszaki igazgatóról, ***Miguel de Icazá***ról ismert. Felismerve néhány, az utóbbi időben napvilágot látott szabványban rejlő lehetőséget, Miguel de Icaza elkészítette azt a programprototípust, amely később a Mono-projektben teljesedett ki.

De mik voltak azok a szabványok, amik szemet szúrtak Miguel de Icazának? Nem titok, hogy ez az ECMA-334 és az ECMA-335 volt, amelyek a Microsoft .NET fejlesztői felület alapját képező technológia leírását tartalmazzák. Ennél a pontnál hangsúlyoznunk kell, hogy lényeges különbség van a .NET fejlesztői felület és az általánosan használt „.NET” kifejezés tartalma között. A Microsoft-termékek és szolgáltatások széles körét – operációs rendszereket, fejlesztői eszközöket, hálózati szolgáltatásokat és alkalmazásokat – illetik a tágabb értelemben használt .NET kifejezéssel. Mi ennek a .NET-nek csak egy bizonyos részével foglalkozunk.

2000 októberében a Microsoft, a Hewlett-Packard és az Intel

Rövid bevezetés a C#-be

A C# magas szintű objektumközpontú programozási nyelv, amelyet a CLI (Common Language Infrastructure, általános nyelvi alap) kiegészítésére terveztek. A tervezői csapat élén *Anders Hejlsberg* (ismert Turbo Pascal- és Delphi-fejlesztő), *Scott Wiltamuth* és *Peter Golde* állt.

Első pillantásra a C# sok mindenben hasonlít a Javára. Bár a parancsformátum szempontjából hasonlóak, a C# számos olyan lehetőséggel is rendelkezik, amivel a Java nem. A szolgáltatások némelyike közvetlenül a C++-ból származik, ilyen a túlterhelés (operator overloading), a felhasználó által meghatározott átalakítások (user-defined conversions), igazi szabályos tömbök és hivatkozás szerinti értékátadás (pass-by-reference). A többi tulajdonság, mint az osztályok, a csatolófelületek, a `struct`, az `enum` és a még különlegesebb `delegate` és függvénymutatók, a C, a C++ és a Java keverékéből származnak.

A korszerű objektumközpontú nyelvek legfontosabb jellemzőinek támogatására a C# szintén rendelkezik a különböző elemszerkezetek – tulajdonságok, eljárások, attribútumok, események és leírás – lehetőségével. Ezzel nagymértékben leegyszerűsödik azoknak a fejlesztőknek a munkája, akik elemeket írnak és RAD-környezetbe akarják foglalni őket.

Azoknak a programozóknak a számára, akik járatosak a Java világában, e kis C#-bevezetésben foglaltak semmilyen gondot nem fognak okozni. Bár az objektumközpontú programozásban gyakorlott programozók érdeklődési területét ez a kis ismertető nem fedi le, a *Kapcsolódó címek* részben a C# nyelv részletesebb leírásaihoz is találhatunk hivatkozásokat.

Bevezetésképpen vessünk egy pillantást a jó öreg Hello World program C#-ben írt változatára:

```
/*
    A hello.cs program kódlistája
*/

using System;

public class HelloWorld
{
    public static void Main()
    {
        for (int x=0; x<3; x++){
            // Console from the System
            // namespace
            Console.WriteLine("Hello World");
        }
    }
}
```

folytatás a következő oldalon

közösen előterjesztette a Common Language Infrastructure (CLI, általános nyelvi alap) nevű futásidejű környezet és egy újonnan kifejlesztett objektumközpontú nyelv, a C# részletes leírását. 2001 második felétől a Ximian hivatalosan is elindította a Mono-projektet, azzal a céllal, hogy létrehozzák a javasolt szabványokon alapuló .NET fejlesztői környezet nyílt forrású megvalósítását. 2001 decemberében az ECMA (European Computer Manufacturing Association, Európai Számítógépgyártók Szövetsége) hivatalosan is jóváhagyta a CLI és C# leírásaira vonatkozó szabványokat.

Áttekintés

A CLI egy alaposztály-eljáráskönyvtárat és egy futtatókörnyezetet jelöl ki, ennek feladata többek közt az azonnali (JIT, Just In Time) fordítást, memóriakezelést, kivételkezelést megvalósító szolgáltatások, valamint a befűzési- és biztonsági szolgáltatások nyújtása. A folyamat jobb szemléltetéséhez segítséget nyújt, ha összehasonlítjuk a forráskód fordításának hagyományos módszerével.

A hagyományos folyamat szerint a forráskódot a fordítóprogram (compiler) alakítja át a gépnek megfelelő utasításokká, amiket közvetlenül a processzor hajt végre. Egy x86-os sorozatú processzor számára lefordított program egy PPC processzoron hibát eredményez, ha a programot előtte a processzornak megfelelően nem fordítjuk újra. Mindez annyiban nehezíti egy program több eszközfelületre való megvalósítását, hogy minden egyes különböző változathoz rendelkeznünk kell a megfelelően fordított kóddal.

A másik lehetőség, hogy egy olyan köztes futtatókörnyezet számára fordítjuk le a forráskódot, amelynek az utasításai függetlenek a háttérben működő eszköztől. A köztes utasítások ezután különböző módszerekkel hajthatók végre. Az egyik módszer, hogy egy értelmezőprogramot (interpreter) használunk. Az értelmezőprogram betölti az utasításokat, majd végrehajtja őket, mintha egy virtuális számítógép működne a háttérben. A második módszer szerint a köztes utasítások a gépnek megfelelő utasításokra való JIT-fordítása futásidőben vagy a telepítéskor történik, s az utasítások ezután közvetlenül hajtódnak végre. Mivel a JIT-fordítás a futtatófelületnek megfelelő utasításokat alkalmaz, a fordítás a célpro-

folytatás az előző oldalról

Minden programnak osztályokból kell felépülnie.

A lista tetején egy megjegyzésrész látható. A C# ugyanazt a megjegyzésformátumot használja, mint a Java vagy a C++. A többsoros magyarázatok használatához a kezdet jelzésére a /* jelet, a végén pedig a */ jelet használjuk. Egysoros megjegyzések esetén a kettős perjel (//) szintén elfogadott.

A megjegyzés alatt nem találunk fejállományt (header), mint a C vagy C++ esetén. Láthatunk viszont egy utasítást, ami a Javából ismert `import` utasításra emlékeztet. A `using` kulcsszó a C#-ben egy névtér megadására alkalmazható, ami lényegében egy osztálygyűjtemény. Ebben az esetben a `System` az az osztálykönyvtár-névtér, amit megadunk. A C# egy kis- és nagybetűket megkülönböztető nyelv, így a `System` nem azonos a `system`-mel. Szintén a Javához hasonló az a tulajdonsága, hogy az utasítások pontosvesszővel zárulnak.

Lefelé haladva a következő sor a legkülső osztály kezdete. A C# fájlonként több osztály megadását is megengedi, így a fájl nevének nem kell a legkülső osztály nevével megegyeznie, mint a Javában; ezzel szemben a fájl nevének *.cs* kiterjesztést kell kapnia.

A `Hello World` osztály nyitó kapcsos zárójel utáni első sora a `Main()` tagfüggvény. Minden programnak rendelkeznie kell `Main()` tagfüggvénnyel, ami a program belépési pontjának a szerepét tölti be. A C++ és Java nyelvektől eltérően a `Main()` nagybetűvel kezdődik. A fő eljárás megadásában szerepel még a `public static void` kifejezés. A `public` beállítja az eljárás láthatóságát és elérhetőségét (lényegében bárki hozzáférhet). A `static` kulcsszó lehetővé teszi, hogy még azelőtt belépjünk az eljárásba, mielőtt egy `Hello World` típusú objektumot hoznánk létre. Végül a `void` kulcsszó azt adja meg, hogy az eljárás nem rendelkezik visszaadott értékkel. A C#-ben a `Main()` tagfüggvény vagy `int` típusú értékkel tér vissza, vagy semmilyen értéket nem ad vissza.

A `Main()` tagfüggvény belsejében az alábbi ciklus helyezkedik el:

```
for (int x=0; x<3; x++){
    // Console from the System namespace
    Console.WriteLine("Hello World");
}
```

A `for` ciklus formátumának a C, C++ vagy Java-programozók számára már ismerősnek kell lennie. A ciklus első része, az `int x=0` egy `x` nevű egész típusú változót ad meg, és a `0` értéket adja neki. Ez a változó tölti be a ciklusszámláló szerepét. A ciklus középső része, az `x<3` kifejezés az a logikai feltétel, ami eldönti, hogy mikor kell a ciklusnak befejeződnie. Esetünkben, ha az `x` értéke `3`-nál nagyobb, a vizsgálat eredménye negatív, kilépünk a ciklusból. A ciklus utolsó része a frissítési szakasz. A C++-hoz hasonlóan a C# is megengedi az utólagos (post) növelést és csökkentést. Az `x++` parancs az `x=x+1` rövidítése. A frissítő szakasz minden ciklusvégrehajtás után lefut. A példában a ciklus 3-szor hajtódik végre.

A C# a `while`, a `do-while`, a `for` és `foreach` típusú ciklusokat támogatja.

A `for` ciklus belsejében a `Console.WriteLine()` tagfüggvény hívása található. A tagfüggvény hivatalos neve `System.Console.WriteLine()`, de mivel kikötöttük, hogy a `System` névteret használjuk, a `System` kezdőrészt elhagyhatjuk. Hosszú programokban ezzel a fogással jelentős mennyiségű gépeléstől kímélhetjük meg magunkat, de ha meggondolatlanul használjuk, bonyolulttá válhat annak a meghatározása, hogy éppen melyik névteret használjuk. A `Console.WriteLine()` tagfüggvény a kimeneti szöveget a rendszerkonzolra irányítja. A példaként vizsgált kód végén lévő kapcsos zárójelek lezárják a ciklust, a tagfüggvényt, majd az osztályt.

A lista például egy *hello.cs* nevű szöveges fájlba menthető, majd a Mono C#-fordítójával, az `mcs`-sel lefordítható. Az eredményként kapott *hello.exe* futtatható fájl az alábbi eredményt szolgáltatja:

```
[jdq@newton mono]$ mcs hello.cs
Compilation succeeded
[jdq@newton mono]$ mono hello.exe
Hello World
Hello World
Hello World
```

Természetesen a C# sokkal többet érdemel annál, mint amit ebben a bevezetőben elmondhattunk róla. A *Kapcsolódó címek* részben néhány jó, a C# nyelvet oktató segédanyagot találhatunk.

cesszornak megfelelően alakítható. A JIT-fordító tovább növelheti a futtás sebességét, oly módon, hogy csak a felhasználásra kerülő utasításokat fordítja le, majd a memóriában tárolja őket a későbbi hívások számára.
A futtatókörnyezet használatának felületfüggetlenségéért a eredményt a futtatási sebesség terén kell köttetnünk. Az eszközfüggő kódra történő fordítás hagyományos módszeréhez képest a végrehajtás lassabb. A sebességkülönbség mértéke függ az adott környezettől és a használt futtatási módszertől. Általában az értelmezőprogram használata igényli a legnagyobb futtatási időt. A JIT-fordítást alkalmazó eljárás sebessége jóval közelebb áll a hagyományos fordítási módszeréhez, mivel mindkettő natív (az adott környezetnek megfelelő) utasításokat eredményez. A futásidejű többletmunka azonban a sebességben mégis eredményez egy kis lemaradást.
Most arra gondolhatunk: egy objektumközpontú nyelv, alapeljáráskönyvtár, futtatókörnyezet – mintha csak a Javáról lenne szó. Ez igaz is – a CLI alkotóelemei nagyon hasonlítanak a Javában lévőkhöz, mégis van egy alapvető különbség. A Java futásidejű része kizárólag a Java nyelv futtatására lett kifejlesztve. Igaz ugyan, hogy néhány más nyelvet is átalakítottak olyan módon, hogy a kimeneti kódja egyezzen a Javáéval, és ezáltal a JVM (Java Virtual Machine, azaz Java virtuális gép) számára futtatható legyen, de ez még mindig nem üti meg a CLI által biztosított nyelvfüggetlenség mértékét. Hiszen a CLI-t éppen arra tervezték, hogy számos programozási nyelv futtatókörnyezete lehessen. A CLI adattípus-rendszere az imperatív nyelvek (mint a C vagy a Pascal) és az objektumközpontú nyelvek támogatására egyaránt képes.
A CLI nemcsak arra képes, hogy többféle nyelven írt programot futtasson (nyelvfüggetlenség), hanem e nyelvek számára az egymás közti adatmegosztáshoz is alapot nyújt (nyelvek együttműködése), ideértve a többnyelvű kivételkezelést is. Az egyik nyelven létrehozott objektumnak a másikban is lehetnek leszármazottai. Ha megvizsgáljuk a CLI alapvető összetevőit, megtudhatjuk, hogy milyen módszerekkel érhető el a nyelvi függetlenségnek ez a magas foka.

A CTS

A CLI lelke a közös típusrendszer, a CTS (Common Type System). A CTS egy osztott típusrendszert határoz meg, amely rögzíti az adatok megadásának, használatának és kezelésének a szabályait. Ennek a szigorúan kötelező típusrendszernek az alkalmazásával a CLI képes biztosítani a típusok épségét, és a nyelvek számára lehetővé teszi a másik nyelv adattípusainak kezelését. A nagyszámú különböző programozási nyelv egyeztethetősége érdekében a CTS két fő adattípust biztosít, amelyek több altípussal, értékkel (értéktípussal) és objektummal (hivatkozástípussal) rendelkeznek. Az értékek az olyan egyszerű adattípusok számára vannak fenntartva, mint az egész és lebegőpontos értékek. Az objektumok a programozási nyelvek számára szükséges összetettebb egyedek leírására használatosak.

A CLS

A CLS (Common Language Specification, közös nyelvleíró) egy olyan keretrendszert ír le, amelyhez a fordítóprogramoknak a nyelvek közti adatcsere érdekében létrehozott programkönyvtárak és bináris állományok előállításakor ragaszkodniuk kell. A CLS valójában a CTS részegysége, ami ésszerű adattípus- és szabályrendszert ad egy nyelv fordítóprogramja számára annak érdekében, hogy az így létrejövő lefordított kódot más nyelvek is képesek legyenek használni, illetve kiegészíteni. Egy nyelv rendelkezhet arról, hogy milyen mértékben támogatja a CLS-t. Azok a nyelvek, amelyek bármely CLS-típus használatát lehetővé teszik, az úgynevezett CLS-fogyasztók. Azok, amelyek képesek CLS-típusok létrehozására illetve kiterjesztésére, a CLS-kiterjesztők. Végül azok a nyelvek, amelyek teljes mértékben magukban foglalják a CLS-t, egyszerre fogyasztók és kiterjesztők.

1. *lista* Részlet egy visszafordított C#-programból a CIL utasításkészlet bemutatására

```
// Eljárás 1 sor
.method public hidebysig specialname
        ➥rtspecialname
        instance default void .ctor() cil
        ➥managed
{
    // Az RVA 0x20ec-nél kezdődő eljárás
    // Kódméret 7 (0x7)
    .maxstack 8
    IL_0000: ldarg.0
    IL_0001: call instance void valuetype /
        ➥[corlib]System.Object::.ctor()
    IL_0006: ret
} // end of method instance default void
  // .ctor()

// Eljárás 2 sor
.method public static
        default void Main()  cil managed
{
    // Az RVA 0x20f4-nél kezdődő eljárás
    .entrypoint
    // Kódrészelet 11 (0xb)
    .maxstack 8
    IL_0000: ldstr "Hello World"
    IL_0005: call void class /

[corlib]System.Console::WriteLine(string)
    IL_000a: ret
} // Vége a void Main() eljárásnak
```

A leíró adat és a CIL

Amikor egy forrásfájlt egy CIL- (Common Intermediate Language, közös közvetítő nyelv) megfelelő fordítóprogrammal fordítunk le, kimenetként egy hordozható futtatható bináris állomány (PE – portable executable) jön létre, amit néha assemblyként is emlegetnek, bár az assembly egynél több fájlból is állhat. A PE két fontos adatrészből tevődik össze. Az első a leíró adat, ami tartalmazza a használt típusok leírását, a CLI által az osztályok előkereséséhez és betöltéséhez használt adatokat, a memóriaelrendezést és egyéb, futásidőben szükséges adatokat. A második rész a közös közvetítő nyelven írt CIL-kód. A CIL a közvetítőutasítások nyelvektől független gyűjteménye. Amikor lefordítunk egy nyelvet a CLI számára, ez a CIL-kód jön létre. A CIL megfelelő teljesítménnyel rendelkezik ahhoz, hogy rengeteg különböző programozási nyelvet kezelhessen, és úgy tervezték, hogy nagy hatékonysággal lehessen a felületnek megfelelő natív utasításokra lefordítani. A CIL-utasítások egy kis része látható az *1. listán*, a Hello World program C#-ben írt változatában.

2. lista A ljlib.cs fájl tartalma

```
using System;

namespace LJlib {

   public class Output
   {
      static public void SayHello ()
      {
         Console.WriteLine("Hello Linux
                        ➥Journal!");
      }
   }
}
```

3. lista A hello.vb fájl tartalma

```
Imports LJlib

Module modmain
   Sub Main()
      Output.SayHello()
   End Sub
End Module
```

A VES

A virtuális futtatórendszer (Virtual Execution System – VES) teremti meg a CLI-re írt programok számára a megfelelő futtatókörnyezetet. Végrehajtja a betöltést, a modulok egyesítését, vezérli a memóriát, megvalósítja a biztonsági szolgáltatások és kivételek kezelését, valamint megfelelő hátteret nyújt a CIL-utasítások végrehajtásához.
A CLI memóriakezelő része tartalmaz egy szemétgyűjtőt (Garbage Collector – GC) is. Más futtatókörnyezetekkel szemben, a CLI megengedi, hogy a forráskódban engedélyezzük vagy tiltsuk a GC használatát. A GC által kezelt adatokat (lefoglalás, felszabadítás) felügyelt (managed) adatoknak nevezzük, ha pedig a GC tiltva van, felügyelet nélküli adatokról beszélünk. A felügyelt kód (a CLI által végrehajtott forráskód) kezelhet felügyelt és felügyelet nélküli adatokat is.

A Mono

A Mono-projekt kettős célt szolgál. Az első, hogy az ECMA CLI- és C#-szabványainak gyakorlati megvalósítását nyújtsa. A második, hogy mindezt a Microsoft .NET fejlesztői felülettel együttműködő módon tegye. Mindkét résznek megvan a maga értéke, és különböző módon szolgálják a Linux javát. Ha például a .NET-megfelelőség megszűnne, a Mono még akkor is a Linux egyik értékes fejlesztői keretrendszere lenne. Mindamellett a Mono azáltal, hogy a linuxos felületet .NET-megfelelővé teszi, futtathatóvá teszi Linuxon a Windows alá fejlesztett programokat. A gondolatmenetet folytatva az is elmondható, hogy a fejlesztők számára immár rendelkezésre fog állni egy megszokott fejlesztői keretrendszer, amely a linuxos programok fejlesztésére történő átállás esetén csökkenti a tanulási kényszerből adódó korlátokat.
A .NET-nek azok a fő részei, amelynek a kiadásán a Mono projekt éppen dolgozik, a Win Forms (System.Windows.Forms), az ADO.NET és az ASP.NET. A Win Forms tartalmaz minden szükséges eljárást, osztályt és eseményt a Microsoft Windows-rendszernek megfelelő grafikus programok kifejlesztéséhez. Mivel a natív Linux grafikus eszközkészlettel szinte lehetetlen a Windows GUI API-hívásokat emulálni, a Mono a WineLib (➲ http://www.winehq.com) segítségével adja a Windows-felületet. Ha már láttunk alkalmazást a Wine alatt futni, akkor tudjuk, hogy a kinézete egyáltalán nem hasonlít a Linux asztali környezeteire. Ennek megoldására a Wine-ban a Mono gondoskodik a témák támogatásáról, hogy ugyanazokat a leképező eljárásokat használja az elemkészletekhez, mint az asztal egyéb részeinél.
Az ADO.NET tartalmazza a Mono számára a .NET adateléréssel kapcsolatos osztályait. Az ADO.NET többre képes egyszerű adatelérésnél: kapcsolat nélküli, méretezhető, XML-en alapuló adatelérési modellt nyújt bármely adatforrásból. Az írás idején körülbelül egy tucatnyi adatbázis működik a Mono ADO.NET adatszolgáltatójaként. A program fejlődéséért, az újabb és újabb gyártó adatbázisának támogatásáért végzett munka folytatódik.
A Monóban az ASPI.NET támogatása két részre lett bontva: a webűrlapokra és webszolgáltatásokra. A webűrlapok alkotják a webalkalmazások felhasználói felületét. A Win Formshoz hasonlóan a webűrlapok is nyújtják a vezérlőeszközökhöz – gombokhoz, szövegdobozokhoz és egyszerűbb vezérlőelemekből álló összetettebb elemekhez – tartozó tulajdonságokat, eljárásokat és eseményeket. Mindez lehetővé teszi, hogy a webes űrlap felülete RAD-eszköz (Rapid Application Development, gyors alkalmazásfejlesztő) segítségével készüljön, a fogd és vidd módszer alkalmazásával, a Gnome Glade programjához hasonlóan. Ezáltal a megjelenítés elválik a program logikájától, csökkentve a szükséges kódolás mennyiségét. A webes szolgáltatások egy SOAP alapú távoli eljáráshívás támogatását kínálják. Az olyan, mindenütt jelen lévő internetes protokollok használatával, mint az XML- és HTTP-szolgáltatások, az adatok és az eredmények hálózatos megosztását teszik lehetővé, még tűzfalakon keresztül is. Az ASP.NET a CLI által támogatott bármelyik nyelven programozható. Ez azt is jelenti, hogy az ASP.NET kódja lefordításra kerül, és nem értelmezőprogram segítségével fut, mint az ASP korábbi változatai vagy egyéb webes parancsnyelvek. Az ASP.NET a Mono számára akár az XSP webkiszolgálón, akár az Apache 2 `mod_mono` összetevőjével elérhető.
A .NET megvalósításában részt vevő Mono-osztálykönyvtárakon túl számos más programkönyvtár és eszköz is érdekes szolgáltatásokat kínál:

- A GTK#, Qt# és Wx.NET a népszerű linuxos grafikus eszközkészletekhez nyújt C#-kapcsolatot. Ezekkel a C#-burkolókkal minden, Monón futtatható nyelv ugyanazokhoz a grafikus eszközökhöz nyer hozzáférést.
- Az OpenGL#, MonoGLo és CsGL a népszerű két- és háromdimenziós API OpenGL-hez ad kapcsolást.
- Az SDL.NET az SDL programkönyvtárhoz történő kötést biztosítja.
- A Gst# Gstreamer multimédiakeretrendszer-kötés.
- Számos kommunikációs programkönyvtár, például a .NET Jabber és a Gnutella.
- NAnt fordítóeszköz (az Anthez hasonló eszköz).

Természetesen ez csak néhány példa, de elég ahhoz, hogy szemléltesse a Monónak azokat a képességeit, amelyek Linux vagy egyéb felületre történő fejlesztés esetén igénybe vehetők.

A Mono használata

Az első lépés a Mono kipróbálásához a ➲ http://www.go-mono.com címen a projekt honlapjának a meglátogatása, ahonnan letölthetjük a forrás tarcsomagjait, vagy a felületünknek megfelelő bináris állományokat. Pillanatnyilag a Monónak Linuxon és Windowson futtatható átirata létezik, de folyamatban van a MacOS X, FreeBSD és más felületekre történő átírás is. Számos Linux-változatra létezik bináris állomány, többek közt Debianra, Red Hatre, SuSE-ra és Mandrake-re. Ha a Ximian Red Carpetet használjuk, a neki megfelelő fájlokat a Mono Chanellen is fellelhetjük. A cikkhez a Mono 0.20-as változatát használtuk. Észrevehetjük majd, hogy a Mono programcsomagokon felül – amelyek a futásidejű, a C# fordító- és osztálykönyvtárakat tartalmazzák – további csemegéket is kapunk. Ezek között találjuk a Mono hibakereső programját, az XSP webkiszolgálót és a Monodoc leírásböngészőt. Amennyiben gondjaink akadnának a Mono telepítésével, forduljunk a honlapon elérhető leírásokhoz.

A Mono jelenleg a következő összetevőkkel érkezik:

- C#- és Basic-fordítóprogramok.
- A VES, ami egy JIT-fordítóból és a hozzá tartozó hulladékgyűjtőből, a biztonsági rendszerből, az osztálybetöltő, ellenőrző és végrehajtó rendszerből áll. Egy értelmezőprogram szintén része ennek az összetevőnek.
- Egy C#-ben írt osztálykönyvtár-gyűjtemény, amely a CLI-szabványban meghatározott osztályokat, a .NET FDL részét képező osztályokat és más Mono által használt osztályokat valósít meg.
- Különböző segédprogramok.

Az `mcs` a Mono C#-fordítóprogramja. Érdekes programozói bravúr, hogy az mcs-t C#-ben írták. A Mono 0.10 óta az `mcs` önmaga fordítására is képes. Ha érdekelnek bennünket az `mcs` parancssori lehetőségei, amelyek megegyeznek a Microsoft C#-fordítója által kínált kapcsolókkal, részletes súgóoldalak állnak rendelkezésünkre.
A Visual Basic.NET Monóban található megfelelőjének, a MonoBasicnek a fordítóprogramja az `mbas`. Bár a fejlesztése még nincs a C#-fordítóéhoz hasonló állapotban, a Basickel való kísérletezgetéshez már megfelelő szolgáltatásokkal rendelkezik.
A Mono két futtatókörnyezete a `mono` és a `mint`. A `mono` a VES CLI meghatározásaival egy JIT-fordítónak megfelelő környezet. Ezzel ellentétben a `mint` egy értelmezőprogram, ami a `mono` számára könnyen hordozható megoldást jelent, de pillanatnyilag csak az x86-os felületet támogatja. A legjobb futási teljesítmény a `mono` használatával érhető el.
A Monóval érkező segédprogramok közül figyelemre érdemes a `monodis` és a `pedump`. A `monodis` egy lefordított assembly visszafejtésére alkalmas, kimenete a megfelelő CIL-kód. Ezt használtuk az *1. listán* látható CIL-kód megjelenítésére. Ha kíváncsiak vagyunk a CIL további részleteire, vagy bepillantást szeretnénk nyerni a hordozható futtatható kód előállításába, játszadozzunk el velük egy kicsit.
Most, hogy már ismerjük a Mono összetevőit, itt az idő, hogy ki is próbáljuk őket. A Mono nyelvi párbeszédének próbájához C#-ben egyetlen eljárással egy egyszerű osztályt írunk, majd egy MonoBasic-programból meghívjuk.
A *2. lista* mutatja a ***ljlib.cs*** C#-könyvtárat, a *3. listán* pedig a ***hello.vb*** MonoBasic-program látható. Az első lépés a ***ljlib.cs*** programkönyvtárrá történő fordítása. A lefordított programkönyvtárak ***.dll*** kiterjesztéssel rendelkeznek, a futtatható állományok kiterjesztése pedig ***.exe***.

- Programkönyvtár fordításához az `mcs -target:library` kapcsolóját használjuk:
  `[jdq@newton]$ mcs -target:library ljlib.cs`
- `Compilation succeeded`
  Ennek eredménye a ***ljlib.dll*** fájl, ami a `Ljlib` névteret és `Output` osztályt tartalmazza. Most a ***hello.vb*** program fordítása következik. Ahhoz, hogy a fordításkor az éppen előállított ***ljlib.dll*** fájl kerüljön felhasználásra, utasítanunk kell a MonoBasicet, hogy hivatkozásként ezt használja. Ezt a `-r` kapcsolóval tehetjük meg:
  `[jdq@newton]$ mbas -r ./ljlib.dll hello.vb`
- `Compilation succeeded`
  Az mbas kimenete a ***hello.exe*** fájl. Ezt a mono segítségével futtathatjuk is:
  `[jdq@newton]$ mono hello.exe`

Hello Linux Journal!

És íme: két nyelv, a C# és a MonoBasic egy futtatható állományban egy időben működik. Bár ez egy végtelenül egyszerű példa, mégis jól mutatja a CLI nyelvfüggetlenségét és együttműködési képességét, és jelzi a Monónak fejlesztőeszközként rendelkezésre álló széles körű lehetőségeit.

Összegzés

Bár a Mono még fejlesztés alatt ál, már így is a Linux programfejlesztése elősegítésének nagy ígérete. Ha az utóbbi két évben mutatott fejlődését vesszük alapul, a Mono jövője nagyon izgalmasnak ígérkezik.

Linux Journal 2003. július, 111. szám

Julio David Quintana (jdq@jdqi.com)
Villamosmérnök. 1997-ben találkozott a Linuxszal, és azóta nem is tud szabadulni tőle. Ha nyelvtani vagy tárgyi hiba kapcsán keressük, akkor nem érhető el, de a dicséreteket és a jókívánságokat szívesen fogadja.

KAPCSOLÓDÓ CÍMEK

A C# összehasonlító áttekintése
➲ http://genamics.com/developer/csharp_comparative.htm

A C# a Java-fejlesztők szemszögéből
➲ http://www.25hoursaday.com/CsharpVsJava.html

A C# Stations C#-oktató
➲ http://www.csharp-station.com/Tutorial.aspx

Az ECMA CLI-szabványa
➲ http://www.ecma-international.org/publications/standards/ECMA-335.HTM

Az ECMA C# nyelvre vonatkozó szabvány
➲ http://www.ecma-international.org/publications/standards/ECMA-334.HTM

A Mono projekt honlapja ➲ http://www.go-mono.com

A Softsteel C#-oktatója és „patchwork” könyve
➲ http://www.softsteel.co.uk/tutorials/cSharp/cIndex.html

Hálózatkezelés a Nagios rendszerrel

A John Deere különféle alkatrészek és programok keverékére volt kénytelen kiterjeszteni a hálózati kezelőrendszert. A nap hőse a Nagios lett.

Amikor elkezdtem dolgozni a John Deere Agricultural Marketing Centerben, 12 állomásunk volt az Egyesült Államok és Kanada különböző pontjain. Ezek a telephelyek eltérő felszereltséggel rendelkeztek: az összes feladatot régi kiszolgálókkal és asztali gépekkel oldották meg – a tartományszolgáltatástól egészen a nyomtatásig. Ez azonban nem igazán egyezett a központosított IT-szervezetről alkotott elképzelésünkkel. Hogyan kezelhetnénk és figyelhetnénk meg a kiszolgálókat és a WAN csatornákat ennyire változatos helyeken? Körülbelül két évvel ezelőtt úgy döntöttünk, hogy a felhasználói adatokat hálózatunk központjából kivisszük a telephelyekre. A kérdés csak az volt, hogy miképpen tudjuk majd szemmel tartani az összes telephely különböző gyártóktól származó eszközeit? Hogy fogalmat alkothassunk a megfigyelendő eszközökről, bemutatunk közülük néhányat: minden fő helyen volt egy Maxtor 4100 hálózatra kötött tárolóeszköz (NAS) kiszolgáló, illetve egy Compaq 1600, ami a nyomtatási és tartományvezérlői feladatokat látta el. Néhány kisebb telephelyen helyi nyomtatókiszolgálóként Dell GX1 asztali gépeket használtunk. A központi állomáson egy Compaq TaskSmart N2400 volt a legfőbb fájlkiszolgálónk.

A változatos alkatrészek következtében egyetlen gyártó eszközvezérlő-készlete sem felelt meg az igényeinknek, az az ötlet pedig, hogy mindent más-más eszközzel figyeljünk meg, nem igazán tetszett. Így aztán gyártófüggetlen megfigyelőeszközök közül kellett választanunk. Az általunk talált megoldásokat megközelítőleg három nagy csoportba sorolhatjuk. A lista aljára azok a megoldások kerültek, amelyek nem igazán voltak képesek megfigyelni a folyamatosan növekvő számú kiszolgálót és asztali gépet. A legtöbb esetben kénytelenek lettünk volna a programmal érkező eszközöknél maradni. Más programok az úgynevezett élvonalbeli eszközök közé tartoztak, ilyen például Hewlett-Packard OpenView rendszere. Ezt ugyan jól használhattuk volna ott, ahol szerettük volna, és képes lett volna megfigyelni is azt, amit akartunk, de az ára messze meghaladta lehetőségeinket.

A NetSaint projekt

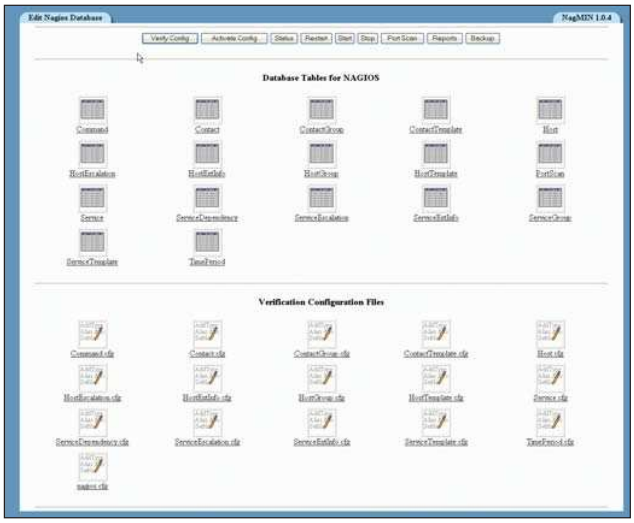
Már éppen kezdtük elveszíteni a reményt, hogy találunk valamit, ami megfelel az igényeinknek, amikor belefutottam a NetSaint projektbe. A NetSaint olyan megfigyelőrendszert ígért, ami megfelelt volna az igényeinknek, és képes lett volna megfigyelni azokat a dolgokat, amiket szerettünk volna, mégpedig egy nyílt keretrendszer segítségével, ami lehetővé tette, hogy saját bővítményeket (plugin) készítsünk. De a NetSaint segítségével nemcsak egyszerűen megfigyelhetjük a kiszolgálókat és a szolgáltatásokat, de azt is lehetővé teszi, hogy a bemutatott irányvonalat követve munkáinkat előretekintőbb módon készítsük el.

Minden nagyszerűen ment: a NetSaint megbízhatóan végezte munkáját a John Deere egyik kisebb irodájában. Ahogy teltmúlt az idő, a mezőgazdasági részleg belekezdett az IT-részleg korszerűsítési programjába, azzal a célkitűzéssel, hogy a szakértői területek színvonalát a teljes részlegben megemeljék. Ebben a projektben ismét felmerültek a megfigyelés régi bonyolódalmai. Mivel egy központosítottabb IT-szerkezet felé próbáltunk meg elmozdulni, a többi megfigyelőprogram valamennyi képességével fel kellett vértezni magunkat. Továbbá minden egység saját maga hozhatta meg az igényeinek a legjobban megfelelő IT-döntéseket. A kérdés tehát az volt: miként fogjuk össze a sokféle megfigyelőrendszert, és hogyan mozduljunk el egyetlen egységes megoldás irányába? Az új megfigyelőnek képesnek kell lennie kiszolgálók százait és szolgáltatások ezreit figyelemmel kísérni, miközben az egyes egységeknek lehetővé teszi, hogy a kialakult irányzatok adatait grafikonon ábrázolják. Ugyancsak nehézséget jelentett az eddigieknél is szórtabb megfigyelendő alkatrészkészlet, a Network Appliance-től kezdve a Sunon át a Dell-gépekig. Továbbra is voltak pénzügyi korlátaink, ez azt jelentette, hogy az OpenView-osztályú termékek kiestek a lehetőségek közül. Mivel termelési környezetünkben a NetSaintet futtattuk, úgy döntöttünk, nekilátunk és megpróbálunk NetSaint alapú megoldást keresni a mezőgazdasági részleg számára. Az első dolog, amit megtanultam, hogyha egy ideig nem követem a nyílt forrás közösségének a történéseit, akkor amikor visszatérek, igen sok változást tapasztalok. Esetünkben a NetSaint nem létezett többé; helyét átadta utódjának, a Nagiosnak (➲ http://www.nagios.org).

1. kép A Nagat-szolgáltatások lapja

Nagios – az utód

A Nagios több szempontból a NetSaint evolúciójának a következő lépcsőfokát jelenti. Némi szomorúsággal szemléltem, hogy a pingvin eltűnt a főoldalról, de lassan hozzászoktam a hiányához. A rendszer tanulmányozása után úgy döntöttünk, hogy a többszintű megoldást választjuk, s a megfigyelési projektünkben szóba kerülő minden egységnél külön Nagios-kiszolgálót helyezünk el. Elsősorban ezért döntöttünk így, hogy egyetlen kiszolgáló se terhelődjön túl, és a szükséges megfigyelési szintet folyamatosan fenntarthassuk. Ezt szem előtt tartva Moline-ban, Illinois államban telepítettük a fő megfigyelőgépet,



2. kép A NagMIN főlapja

majd az első gyermekkiszolgálót a kansasi Lenexában, a mezőgazdasági részleg marketingosztályán helyeztük el.

Telepítés

A szülő- és a gyermekkiszolgálón ugyanaz a Nagios-telepítés futott. Mindkét gépre Red Hat 8-at telepítettünk, majd felpakoltuk a Nagios rendszert. A rendszerrel érkező leírás csaknem teljes értékű, és szépen végigvezet bennünket a telepítés lépésein. A Nagios mellett minden kiszolgálóra egy Nagat nevű programot is feltelepítettünk (*1. kép*).

A Nagat beállítóprogram

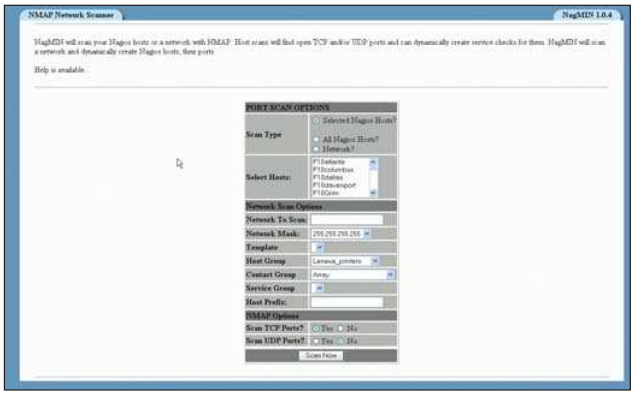
A Nagat a Nagioshoz szánt webalapú beállítóprogram. Segítségével mindössze egy weboldalon kell néhány mezőt kitöltögetnünk, és a válaszaink alapján elkészül a beállításfájl. Így egyetlen terminálablak megnyitása nélkül a kevésbé tapasztaltak is be tudják állítani a Nagiost, illetve annak szolgáltatásait és gépeit. A Nagat használata során Red Hat 8-as telepítésünk alatt belefutottunk néhány nehézségbe. Megfelelőségi gondjaink elhárítására a PHP és az Apache egyik korábbi változatát kellett feltelepítenünk (PHP 4.1.2, Apache 1.3). Miután ezt megoldottuk, hamar be tudtuk indítani a rendszert. A Nagat ugyan néhány hibát is magában rejtett, de könnyen kiküszöböltük őket. Az egyik hiba a ***service edit*** oldalon bukkan fel: a rendszer nem menti a kapcsolatcsoportot (contact). A javításhoz mindössze annyit kell tennünk, hogy az alábbi sorokat a 39. sor `case`-szerkezetéhez illesztjük:

```
$saveobject[´contact_groups´] =
   @implode("´",$saveobject[´contact_groups´]);
```

Ezek után kapcsolatcsoportunkba már gond nélkül kimenthetjük a változásokat, a szolgáltatás frissítése közben.

A kiszolgálók beállítása

Miután mindent feltelepítettünk, elérkeztünk a projekt – véleményem szerint – legnehezebb részéhez: a kiszolgálók beállításához. Végül maga a kiszolgálóbeállítás közel sem volt olyan nehéz feladat, mint kitalálni, hogy mit is várunk el rendszerünktől, majd végigjárni az odáig vezető utat. Az első döntés, amit meg kellett hoznunk – és amit éppen most vonunk vissza –, a Nagios fordítása beépített adatbázis-kezelés nélkül.



3. kép A NagMIN kapupásztázó felkutatja a megfigyelendő szolgáltatásokat

Egy teljes évig meg is felelt az igényeinknek, de ahogy újabb egységeket építettünk be Nagios megfigyelő rendszerükbe, az általa nyújtott adatok elérése egyre fontosabbá vált. Többé már nem voltak elégségesek a Nagios saját kis grafikonjai. Magunk szerettük volna az adatokat megkapni és megjeleníteni, emberek és egységek szerint, ezt pedig nem éppen egyszerű megtenni az eredeti Nagios-telepítésünk által adott egyszerű szöveges állományokkal.

A Nagios beállítása

A következő lépést a főrendszer beállítása jelentette, ami a legtöbb szempontból magától értetődő volt. Akad néhány dolog, amit érdemes egy kicsit jobban megnézegetni, ha a legtöbbet szeretnénk Nagios rendszerünkből kihozni. Az egyik ilyen képesség az intelligens ellenőrzés (smart check) módszere. Ahelyett, hogy rendszerünkön az összes ellenőrzést egyszerre futtatná (ez igen nagy processzorfelhasználással járna), a Nagios szétosztja az ellenőrzést, mondjuk ötperces időközökre, jelentősen mérsékelve ezáltal a CPU-felhasználást. A másik dolog, amit érdemes megnézni: a párhuzamosított szolgáltatásellenőrzés; ezáltal egy időben több ellenőrzést is futtathatunk, ami sokat segít, ha több processzorral rendelkezünk.
A szülő- és a gyerekkiszolgáló beállítása szinte teljesen megegyezik, akad azonban két lényeges különbség is. A gyermekkiszolgálón alapértelmezés szerint a figyelmeztetéseket ki kell kapcsolni, illetve be kell állítani az aktív ellenőrzést (active check), míg a szülőkiszolgálón ennek éppen az ellenkezőjét kell végrehajtanunk. Mindezt azért tesszük, hogy csökkentsük a terhelést a főkiszolgálón és az összes figyelmeztetést egyetlen gépen jelenítsük meg. (A kétféle kiszolgálóhoz használható beállításfájlok az 51. CD Magazin/NAGIOS könyvtárában találhatóak.)
A következő lépés a Nagios által végrehajtandó parancsok megadása lesz. Kísérletet tettem arra, hogy a rendszert használó emberek eszébe véssem, hogy a Nagios: keretrendszer. Ez alatt azt értem, hogy ő maga semmilyen ellenőrzést nem hajt végre, ez a munka a Nagios által meghívható bővítményekre marad. És ez nagyon jó így, hiszen ezáltal könnyedén készíthetünk saját bővítményeket, mindaddig, amíg ragaszkodunk a Nagios nyújtotta keretrendszerhez. Az általunk írt bővítmények segítségével olyan rendszereket is képesek leszünk megfigyelni, amiket az eredeti Nagios bővítményeivel nem tudtunk volna. Jelenleg képesek vagyunk megfigyelni például a NetApp fájlkezelőt (filer), és kiolvasni a lapszámlálót a HP nyomtatókból, valamint a Compaq Insight Manager adatait be tudjuk építeni a Nagios rendszerbe.
A Microsoft Windows-kiszolgálók megfigyelésének talán

4. kép Nagios-szolgáltatásrészletek

legegyszerűbb (de nem az egyetlen) módja, ha kihasználjuk az NSClient képességeit. Az NSClient programot windowsos gépeinken szolgáltatásként futtassuk. Futtatásához nyissunk meg egy kaput a Windows-kiszolgálón – biztonsági okokból érdemes jelszóval védeni a kapuelérést. Írásom születésekor még nem lehetett titkosítást használni a program és a Nagios kiszolgáló közötti kapcsolatban. A bővítmény lehetővé teszi, hogy elérjük a kiszolgáló memória- és lemezfoglaltságát, a processzorteljesítményt, illetve egyéb olyan adatokat, amiket Windows alatt egyébként a teljesítménykezelő (performance manager) eszköz segítségével érhetnénk el.

Linux- és Unix-rendszereinkhez némileg eltérő módszert használunk. A Nagios Service Check Acceptor (NSCA) nevű programot démonként futtatjuk az `inetd` alól. A program használatához a Nagios egy `check_nsca` nevű bővítményt indít el, ami titkosított csatornát hoz létre a két számítógép között, majd a bővítmény segítségével lefuttatja a számítógép ellenőrzéséhez szükséges kódokat. A rendszer beállítása magától értetődő: válasszuk ki a két gép között használható titkosítási módszert, majd az ügyfélen adjuk meg a `plugin` parancsot. Ezt követően az adatokat máris átszipkázhatjuk Nagios rendszerünkre.

Mint azt korábban említettem, jelenleg éppen azzal foglalkozunk, hogy a Nagios által alapértelmezés szerint használt fájlokat háttéradatbázisba töltsük át. Ezt a módszert arra szeretnénk használni, hogy jobb adatábrázolást érjünk el olyan eszközök segítségével, mint például a Crystal Reports. A Nagios önmagában is támogat két adatbázis-kezelőt: a MySQL-t és PostgreSQL-t. A Nagios rendszerrel együtt néhány parancsfájlt kapunk, amelyek előállítják az adatbázist, és elkészítik Nagios rendszer működéséhez szükséges táblákat és mezőket. Mi a PostgreSQL mellett döntöttünk. Az adatbázis felállítása után a `--with-mysql-xdata` vagy a `--with-pgsql-xdata` kapcsolók valamelyikével újra futtatnunk kell a beállítási parancsfájlokat. Az `xdata` úgy állítja be a rendszert, hogy mindenhez adatbázisokat használjon. Jelenleg az egyetlen dolog, ami nem támogatja az adatbázisokat, maga a beállításfájl.

Végül egy érdekességre szeretném felhívni a figyelmet: idén márciusban jött ki egy kiegészítés, mégpedig Nagios rendszer NagMIN elnevezésű Webmin bővítménye (*2. kép*). Ez a rendszer nemcsak a Nagios webes beállítását teszi lehetővé (mint a Nagat), hanem néhány olyan dolgot is felkínál, amit eddig egyetlen eszköz sem. Először is lehetővé teszi a Nagios-beállításfájlok adatbázis-támogatását. Másodszor, ha a Nagios-telepítésünket akarjuk beállítani, igencsak jól jön kapuvizsgálat (port scan) képessége (*3. kép*). Még nem próbáltam ki e képesség minden lehetőségét, de állítása szerint hálózati felderítést végez, és a fellelt adatokat beilleszti a Nagios beállításfájljaiba, megkímélve bennünket attól, hogy a rengeteg adatot mind nekünk kelljen felvennünk. A NagMIN a legtöbb rendszeren jól működik, de a különleges rendszereket valószínűleg mindenképpen kézzel kell felvennünk a rendszerbe. Amennyiben további tájékoztatásra lenne szükségünk a bővítménnyel kapcsolatban, látogassunk el a SourceForge.net honlapjára.

Nagios kiszolgálónkat a szülő-gyermek viszony alapján állítottuk be. Ehhez némileg eltérő beállítások szükségesek, mintha egyszerű önálló rendszert készítenénk. Egyrészről a webfelületet csak a szülőkiszolgálóra telepítettük fel (*4. kép*). A gyermekkiszolgálók mindössze annyit tesznek, hogy ellenőrzési adataikat passzív ellenőrzési módszerrel egyszerűen áttöltik a szülőkiszolgálóra. Ennek kivitelezéséhez egy OCSP parancsot adtunk ki, ami lefuttatja az adatokat a Nagios szülőkiszolgálóra közvetítő parancsfájlt. Azáltal, hogy a parancsfájl futtatására az OCSP parancsot használjuk, minden egyes Nagios-ellenőrzési folyamat után le fog futni. Éppen ezért tulajdonképpen csak a gyermekkiszolgálóknak kell aktív ellenőrzéseket futtatniuk. Így a Nagios-szülőkiszolgálónak csak a webfelületet kell futtatnia, illetve hiba esetén elküldenie a jelentést.

Most, hogy körülbelüli képet kaphattunk arról, hogyan állítottuk be a Nagiost, engedjék meg, hogy elmeséljem, milyen előnyökkel járt a használata. Ennek az évnek az elején vásároltunk egy TempTrax nevű digitális hőmérőt, ami együttműködött a Nagiosszal. A hőmérőt fő számítógéptermünk hőmérsékletének a megfigyelésére használtuk. Március eleje környékén aztán kiderült, milyen fontos is számunkra a Nagios. Egyik pénteken körülbelül déltájt üzenetet kaptam, miszerint a számítógépterem hőmérséklete emelkedik. Miközben öltözködtem, a rendszer válságos helyzetről szóló üzeneteket kezdett nekem küldözgetni a számítógépszoba hőmérsékletével kapcsolatban. Mire beértem a munkahelyemre, a hőmérséklet 80 fok körül volt, és egyre emelkedett. A Nagiosnak hála elég időm maradt, és fel tudtam hívni a légkondicionálásért felelős szakembereket, akik megjavították a berendezést, még mielőtt a számítógépeket komoly kár érte volna. Később kiderült, hogy a számítógépszoba A/C egységét vezérlő másik két rendszer képtelen volt kitárcsázni és riadóztatni a rendszert felügyelő szakembereket. Ha a Nagios nem figyelmeztetett volna bennünket a számítógépszobában előállt hibára, másnap reggel sokkal nagyobb bajjal találtam volna szemben magam.

Egészében véve azt kell mondanom, hogy a Nagios kiváló hálózati megfigyelőeszköz. Keretrendszerről van szó, ami önmagában nem túl sok mindenre képes, de pontosan ez az, ami olyannyira nagyszerű eszközzé teszi. Mivel keretrendszer, csak azt végzi el, amire szükségünk van. Lévén nyílt, a bővítményeket tervezni hozzá éppolyan egyszerű, mint ellenőrzési adatainknak Nagios által is érthető formátummá alakítása. Így a bővítményünk mindent felhasználhat, amit csak a Nagios nyújtani tud, és ez nem kevés. Végül a Nagios olyasvalami, amire mindenképpen vetnünk kell legalább egy pillantást – helyi telepítésünkön akár ki is próbálhatjuk. Kötve hiszem, hogy kiábrándulunk belőle.

Linux Journal 2003. július, 111. szám

Richard C. Harlan
(harlanrichardc@johndeere.com)
Hálózati mérnök a lenexai John Deere Agricultural Marketing Centerben, Kansasban.

Hogyan indexeljünk?

Honlapunkon valószínűleg akad valamilyen keresőfelület – de mi a helyzet a súgóoldalak vagy a levelezőrendszer keresésével?

Számtalan oka lehet annak, ha indexeket szeretnénk rendelni a dokumentumokhoz. Az egyik ilyen gyakran emlegetett ok a honlapok keresőfelülete, de ugyanígy elképzelhető, hogy valaki a levelezését vagy a műszaki dokumentumait szeretné indexelni. Aki próbált már ilyen rendszert készíteni, az valószínűleg jól tudja, hogy egyáltalán nem olyan egyszerű dolog, mint amilyennek első látásra gondolnánk. Számos esemény jöhet közbe, ami megakadályozza a hatékony munkát.

A vénséges vén és nélkülözhetetlen `grep` és társai igen hatékonyan keresnek szöveges adatokban. A `grep`, az `egrep` és rokonaik azonban nem tudnak bármit megadni nekünk. Nem keresnek több soron keresztül, nem tudnak rangsorolt keresési eredményt megjeleníteni, és lineáris keresési algoritmusuk nem igazán teszi alkalmassá őket a nagyobb dokumentumokkal való munkára.

A HTML sem lendít sokat a dolgon. Megjelenítés-központú képességei, egyedi nyelvezete, a formázási és entitástagok serege igencsak megnehezíti a helyes értelmezést.

Az adattárolási skála másik végén az adatbázisokba szervezett adatok állnak. Mindenhol megtalálható példa erre az SQL adatbázis, amely viszonylag kifinomult keresési eszközökkel rendelkezik, de többnyire nem fut igazán gyorsan. Néhány adatbázismotor – kiemelten a MySQL 4 – a kérdést gyors és rendszerezett (ranked) kereséssel próbálja megoldani, viszont nem lehet a kívánt mértékben testreszabni.

Ebből a cikkből megtudhatjuk, miképpen készíthetjük el a saját indexeinket Linux alatt a SWISH-E, a Perl és az XML segítségével. A példákon keresztül bemutatjuk, hogyan használhatjuk a SWISH-E programot HTML-fájlok, PDF-fájlok és súgóoldalak indexének a felépítéséhez.

A SWISH-E (Simple Web Indexing System for Humans – Enhanced) az 1994-ben ***Kevin Hughes*** készítette SWISS utódja. A SWISH 1996-ban hibajavítás és új képességek hozzáadása céljából átkerült az UC Berkeley könyvtárba. Az eredményt GPL alatt adták ki, és SWISH-E-re nevezték át. A fejlesztések ***Bill Moseley***, a jelenlegi projektkarbantartó vezénylete mellett tovább folytatódnak, Billt fejlesztők egész csoportja segít a munkájában.

Mi a SkateboardDirectory.com-nál akkor akadtunk rá a SWISH-E-re, amikor indexelő eszközkészletet kerestünk. Rájöttünk, hogy egyedülálló képességekkel bír: a SWISH-E nemcsak gyors és erőteljes eszközkészletet kínál, amivel indexeket építhetünk, illetve kérdezhetünk le, de egyben kiválóan dokumentált, folyamatos fejlesztés és hibajavítás alatt áll, és Perl csatolófelülettel is rendelkezik. Az is nagyon tetszett, hogy a csomagfelelős Moseley és a többi tapasztalt SWISH-E-felhasználó és -fejlesztő általában igen gyors és pontos választ ad, ha a SWISH-E-levelezőlistán kérdések vagy hibák merülnek fel.

A SWISH-E telepítése

Példánkat egy gyári Red Hat 7.3 munkaállomáson futtattuk, amire a Software Development csomagcsoportot telepítettük fel. A példákat Red Hat 6.2-t futtató munkaállomáson, valamint Debian Woody alatt is kipróbáltuk.

Jelenleg Red Hat alatt csak forrásból rakhatjuk fel a SWISH-E rendszert, továbbá felépítéséhez előzőleg a ***zlib*** és ***libxml2*** könyvtárakat is fel kell telepítenünk. Ha úgy tűnik, hogy valamelyiket fel kell raknunk, a terjesztésünk csomagjai között valószínűleg megtaláljuk. Példáinkban az `xpdf` csomagot is használni fogjuk, így ezt is érdemes feltelepíteni, ha esetleg még nem tettük volna meg. Az általunk megjelölt Red Hat 7.3 munkaállomás az összes SWISH-E rendszerhez szükséges valamennyi előtelepítési feltételt teljesíti.

A következőkben bemutatjuk a SWISH-E 2.4 használatát, ami a fejlesztőcsapat szerint körülbelül a cikk megjelenésével egy időben lesz majd elérhető. A SWISH-E rendszert a következő parancsok sorozatával telepíthetjük (az x.x helyére az időszerű változatszám kerül):

```
% wget http://swish-e.org/Download/
➥swish-e-x.x.tar.gz
% tar zxf swish-e-x.x.tar.gz
% cd swish-e-x.x
% ./configure
% make
% make test
```

Ha a SWISH-E futtatható állományt, a C-könyvtárakat és a súgóoldalakat az alapértelmezett ***/usr/local*** könyvtárba szeretnénk helyezni, rendszergazdaként gépeljük be a `make install` parancsot. Így a SWISH-E végrehajtható állomány a ***/usr/local/bin*** könyvtárba kerül. Amennyiben ez a könyvtár még nincs a `PATH` útvonalunkban, két dolgot tehetünk: vagy módosítjuk a megfelelő ponttal kezdődő fájlt, hogy a ***/usr/local/bin*** könyvtárat is helyezze a `PATH` változóba, vagy teljes útvonalával együtt hívjuk meg a `swish-e` végrehajtható állományt: ***/usr/local/bin/swish-e***.

Most készítsük el és telepítsük a forrás Perl-könyvtárában található `SWISH::API` Perl-modult. Később még szükségünk lesz rá, amikor a súgóoldalaink indexeihez használt Perl-ügyfelet készítjük el. A `SWISH::API` modult a szokásos Perl modultelepítési folyamattal tehetjük fel:

```
% cd perl
% perl Makefile.PL
% make
% make test
```

Ezt követően a `make install` parancsot rendszergazdaként begépelve telepíthetjük a `SWISH-E` Perl-modult.

Miután a `SWISH-E` és a `SWISH::API` Perl-modult teljesen feltelepítettük, a SWISH-E kipróbálásához a HTML-fájlok nyilvántartására készítsünk egy egyszerű indexet. Ebben a példában az Linux Documentation Project (LDP) HOGYAN-

1. lista Az sman-index-prog.pl az indexeléshez súgóoldalakat alakít XML formátumúvá

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

use strict;
use File::Find;

my ($cnt, @files) = (0, get_man_files());
warn scalar @files, " man pages to
                    ➥index...\n";
for my $f (@files) {
   warn "processing $cnt\n" unless ++$cnt % 20;
   my ($hashref) = parse_man($f);
   my $xml = make_xml($hashref);
   my $size = length $xml;
   print "Path-Name: $f\n",
       "Document-Type: XML*\n",
       "Content-Length: $size\n\n", $xml;
}

sub get_man_files {  # angol kézikönyvoldalak
                     # keresése
    my @files;
    chomp(my $man_path = $ENV{MANPATH} ||
      `manpath` || ´/usr/share/man´);
    find( sub {
      my $n = $File::Find::name;
      push @files, $n
      if -f $n && $n =~ m!man/man.*\.!
   }, split /:/, $man_path );
   return @files;
}
sub make_xml { # a tömb XML-változat kiírása
   my ($metas) = @_;
   my $xml = join ("\n",
   map { "<$_>" . escape($metas->{$_}) .
         ➥"</$_>" }
   keys %$metas);
   my $pre = qq{<?xml version="1.0"?>\n};
   return qq{$pre<all>$xml</all>\n};
}
sub escape { # az átadott számokat módosítja!
   return "" unless defined($_[0]);
   s/&/&amp;/g, s/</&lt;/g, s/>/&gt;/g
   for $_[0];
   return $_[0];
}

sub parse_man {    # ez a lényeg
   my ($file) = @_;
   my ($manpage, $cur_content) = (´´, ´´);
   my ($cur_section,%h) = qw(NOSECTION);
   open FH, "man $file  | col -b |"
   or die "Failed to run man: $!";
   my ($line1, $lineM) =
      ➥(scalar(<FH>) || "", "");
   while ( <FH> ) {  # kézikönyvoldalak
                     # szakaszolása
      $line1 = $_ if $line1 =~ /^\s*$/;
      $manpage .= $lineM = $_ unless /^\s*$/;
      if (s/^(\w(\s|\w)+)//
         ➥|| s/^\s*(NAME)/$1/i){
         chomp( my $sec = $1 );
         $h{$cur_section} .= $cur_content;
         $cur_content = "";
         $cur_section = $sec; # új szakasz név
      }
      $cur_content .= $_ unless /^\s*$/;
   }
   $h{$cur_section} .= $cur_content;

   # megvizsgálandó a NAME, HEAD, FOOT,
   # (és esetleg a fájlnév is).
   close(FH) or die "Failed close on pipe
                    ➥to man";
   @h{qw(A_AHEAD A_BFOOT)} = ($line1, $lineM);
   my ($mn, $ms, $md) = ("","","","");
   # NAME mn, DESCRIPTION md, & SECTION ms
   for(sort keys(%h)) { # először A_AHEAD  #
                        & A_BFOOT
      my ($k, $v) = ($_, $h{$_}); # key&val
                                  # másolása
      if (/^A_(AHEAD|BFOOT)$/) {
          # look for the ´section´ in ()´s
         if ($v =~ /\(([^)]+)\)\s*$/)
            ➥{$ms||= $1;}
            } elsif($k =~
            ➥s/^\s*(NOSECTION|NAME)\s*//) {
         my $namestr = $v || $k;
         if ($namestr =
            ➥~ /(\S.*)\s+--?\s*(.*)/) {
            $mn ||= $1 || "";
            $md ||= $2 || "";
         } else { # ez a regex hibázhat.
            $md ||= $namestr || $v;
         }
      }
   }
   if (!$ms && $file =~ m!/man/man([^/]*)/!) {
      $ms = $1; # a  sec-et a path-ból
                # vesszük ha nem található
   }
    ($mn = $file) =~ s!(^.*/)|(\.gz$)!!
    ➥unless $mn;
    my %metas;
    @metas{qw(swishtitle sec desc page)} =
       ($mn, $ms, $md, $manpage);
    return ( \%metas ); # ref visszaadása 5-
                        # kulcsos tömbnek.
}
```

jainak egy oldal/fejezet alapú változatának HTML-oldalait fogjuk indexelni, ami a *~/HOWTO-htmls/* könyvtárba kerül. A cikkben felhasznált LDP-dokumentumok a ➲ http://www.tldp.org/docs.html oldalról származnak.

HTML indexelése a fájlrendszeren

A SWISH-E alapú index létrehozásának első lépése a beállításfájl elkészítése. Hozzunk létre egy *~/indices* nevű könyvtárat, majd lépjünk bele, és a következőket írjuk be egy

./howto-html.conf nevű állományba:

```
# howto-html.conf
IndexDir  ../HOWTO-htmls/

IndexOnly .html

IndexFile ./howto-html.index
```

Az `IndexDir` parancs adja meg azt a könyvtárat, ahol a `SWISH-` az indexelendő fájlokat keresi. Az `IndexOnly` utasítás jelzi, hogy csak a *.html* végződésű állományokat kell indexelni. Végül a létrehozandó index helyét a `IndexFile` utasítás adja meg.

Az első indexünk

A következő paranccsal készítsük el HTML-fájlindexünket:

```
% swish-e -c howto-html.conf
```

A `-c` kapcsoló adja meg, hogy melyik SWISH-E beállításfájlt kell felhasználni. Régebbi rendszereken az index elkészítése akár perceket is igénybe vett, egy mai gépen azonban egy perc alatt végezni kell. Az *1. ábra* a HTML-fájlok SWISH-E alapú indexelésének folyamatát mutatja be a fájlrendszeren.

Keresés az indexben

Próbáljuk ki az új indexünket, és végezzünk el egy egyszerű keresést, ami megadja az NFS-kifejezéssel kapcsolatos HTML-fájlokat. A SWISH-E indexeket a `swish-e` végrehajtható állomány segítségével gyorsan és könnyen tesztelhetjük, ha az indexet a `-f` kapcsolóval adjuk meg, majd a keresett szöveget a `-w` kapcsoló után írjuk. A SWISH-E indexek keresései kis- és nagybetűérzékenyek. Mivel igen sok oldalt (vagy találatot) várunk, ami tartalmazza az NFS szót, a `-m 3` kapcsolóval háromra korlátozzuk a keresések számát:

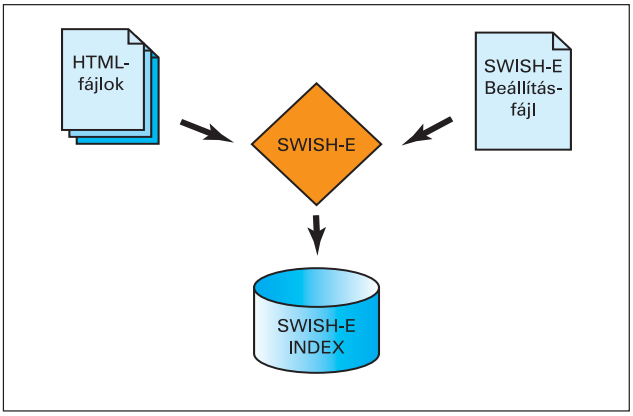
```
% swish-e -f howto-html.index -m 3 -w nfs
```

Az előző sor a következőket adja vissza (rövidítve és újraformázva):

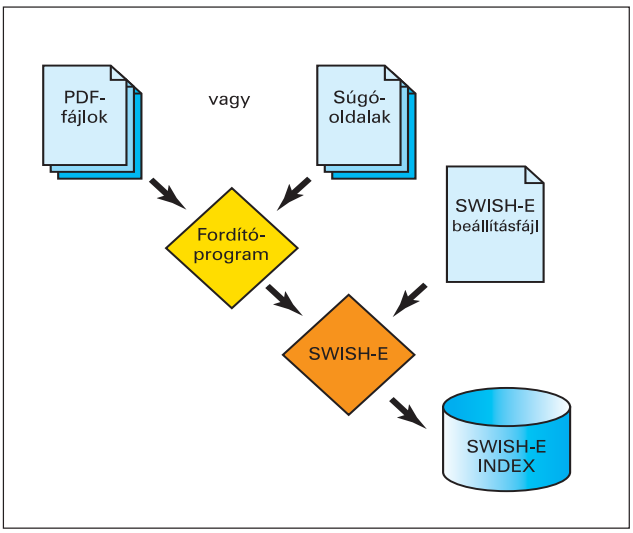
```
1000 ../HOWTO-htmls/NFS-HOWTO/performance.html
       "Optimizing NFS Performance" 33288
998 ../HOWTO-htmls/NFS-HOWTO/intro.html
       "Introduction" 10966
993 ../HOWTO-htmls/NFS-HOWTO/security.html
       "Security and NFS" 35968
```

Nem rossz – ezek az oldalak egyértelműen az NFS-ről szólnak, és a kimenet is intuitív. Az első oszlop a SWISH-E által adott helyezés. A leginkább ide tartozó találatok mindig 1000-es helyezést kapnak, míg a kevésbé ide tartozóak egyre kisebb rangot. A második oszlop a fájlnevet mutatja, a harmadik az oldal címét adja meg, míg a negyedik az indexelt adat bájtszámlálója. A SWISH-E egyik HTML-értelmező motorjának segítségével a HTML-tagokból minden egyes oldal címét meghatározza.

A beépített SWISH-E értelmező motorok neve TXT, HTML és XML. Valamennyi értelmező a nevének megfelelő adattípus kezelésére képes. A SWISH-E jelenlegi változatai már képesek a *libxml2* könyvtár HTML2 és XML2 értelmező hátterét is használni. A beépített változatokkal szemben inkább az XML2 és a HTML2 értelmező használatát javaslom, különösen igaz ez a HTML2 esetében. Ez az oka annak, hogy a *libxml2*, bár



1. ábra HTML indexelése a fájlrendszeren SWISH-E rendszerrel



2. ábra Tetszőleges adatok indexelése külső programmal és SWISH-E-vel

technikailag csak javasolt rész a SWISH-E létrehozásakor, majdhogynem előkövetelmények számít.

A SWISH-E keresési formátumának alapjai

A SWISH-E teljes értékű szövegkereső nyelvvel rendelkezik, amelynek formátumában megtalálható az `AND`, `OR`, `NOT` logikai szerkezet és a zárójelezés. Valamennyi az elvárásoknak megfelelően működik. Például a következő kereséseknek logikus formátumú a szerkezete:

```
% swish-e -f howto-html.index -w nfs AND tcp
% swish-e -f howto-html.index -w nfs OR tcp
% swish-e -f howto-html.index -w '(gandalf OR
frodo) OR (lord AND rings)'
```

A beállításfájl

A SWISH-E beállításfájljai egyszerű szöveges állományok, amelyekben minden egyes sor parancs vagy megjegyzés lehet. Azokat a sorokat, amelyekben az első nem üres karakter a # (kettős kereszt), a SWISH-E megjegyzésként figyelmen kívül hagyja. Minden más nem üres sornak a következő formátumúnak kell lennie:

```
Utasítás kapcsoló [érték] ...
```

Amennyiben üres karaktereket tartalmazó kapcsolót kell megadnunk, idézőjeleket használunk:

```
Utasítás "kapcsoló üres karakterekkel!"
```

Amennyiben a kapcsoló egyszeres idézőjelet (aposztrófot) tartalmaz, a kettős idézőjelet használjuk, és vice versa, például:

```
Utasítás "Fred´s Index Option"
Utasítás ´Josh "joshr" Rabinowitz alkotása´
```

(és ha a szöveg mindkét fajta idézőjelet tartalmazza, akkor sajnáljuk: – a ford.)

A SWISH-E beállításfájljaiban tucatnyi parancsot adhatunk meg, a SWISH-E leírásában kimerítő listát találunk róluk.

Az index

Minden SWISH-E index egy fájlpárban tárolódik. Az egyik fájl neve az `IndexFile` utasítás alapján készül, a másik neve mindig ***indexname.prop*** lesz. Ha SWISH-E indexről beszélünk, mindig erre a fájlpárra gondolunk.
Az indexek elég nagyra nőhetnek. Az iménti HTML-fájlindexelő példánkban használt index 11 MB-ot foglalt el – ez körülbelül az egynegyede az eredeti, beindexelt állományok méretének.

PDF-állományok indexelése

Egészen mostanáig csak a HTML-, XML- és szövegfájlok indexeléséről beszéltünk. Nézzünk meg egy haladóbb példát: indexeljünk be a Linux Documentation Project PDF-dokumentációit.
Ahhoz, hogy a SWISH-E tetszőleges (PDF- vagy egyéb) fájlokat indexelhessen, az állományokat előbb szöveges fájlokká kell alakítani, lehetőleg HTML vagy XML alakúra, majd a SWISH-E segítségével indexelnünk kell az eredményt.
A PDF-fájlokat tehát úgy indexelhetjük, ha a lemezen mindegyiket átalakítjuk, majd indexeljük őket. Ehelyett azonban inkább megragadjuk az alkalmat, és egy rugalmasabb indexelési megoldást mutatunk be: a SWISH-E programozott elérési módszerét (2. *ábra*).
A PDF-állományok indexelését a SWISH-E beállításfájl létrehozásával kezdjük. Nevezzük el ***howto-pdf.conf***-nak, majd a következő tartalommal töltsük fel:

```
# howto-pdf.conf
IndexDir       ./howto-pdf-prog.pl
                 # prog file to hand us XML
                 # docs
IndexFile      ./howto-pdf.index
                 # Index to create.
UseStemming    yes
MetaNames      swishtitle swishdocpath
```

Itt az `IndexDir` utasítás most azt jelenti, hogy a SWISH-E e külső program meghívásával kapja meg az indexelendő állományok adatait, és nem a könyvtárban található fájlokat használja. A `UseStemming yes` utasítás hatására a SWISH-E indexelés vagy keresés előtt kikeresi a szavak tövét. E nélkül a szolgáltatás nélkül a „runs” szó keresésekor a „running” szót tartalmazó dokumentumokat nem találná meg. A szótőkereséssel a SWISH-E felismeri hogy a „runs” és „running” szavak töve azonos, és megtalálja a megfelelő dokumentumokat.

Beállításfájlunk utolsó, de nem kevéssé fontos bejegyzése a `MetaNames` utasítás. Ez a sor különleges képességgel ruházza fel az indexünket: lehetőségünk nyílik csak a címekben vagy fájlnevekben keresni.
Írjuk meg az indexelendő PDF-fájlokról adatokat visszaadó külső programot! Hagyományosan a SWISH-E forrásával együtt egy ***pdf2xml.pm*** nevű példamodult is kapunk, ami az xpdf csomagot használva PDF-fájlokat alakít át XML formátumúvá, megjelölve őket a SWISH-E által használható előtagokkal. Ezt a modult (a ***~/indices*** könyvtárba másolva) a ***howto-pdf-prog.pl*** nevű külső programunkban felhasználhatjuk:

```
#!/usr/bin/perl -w
use pdf2xml;
my @files =
    `find ../HOWTO-pdfs/ -name ´*.pdf´
     ➥-print`;
for (@files) {
    chomp();
    my $xml_record_ref = pdf2xml($_);
    # Ez egy XML fájl
    # SWISH-E fejléc
    print $$xml_record_ref;
}
```

Miután a SWISH-E beállításfájllal és a fenti külső programmal felvérteztük magunkat, készítsük el az indexet:

```
% swish-e -c howto-pdf.conf -S prog
```

A `-S prog` kapcsoló mutatja meg a SWISH-E-nek, hogy az `IndexDir` valójában egy program, ami az indexelendő adatokról ad vissza információkat. Amennyiben a `-S prog` kapcsolót elfelejtjük megadni, miközben külső programot használunk a SWISH-E-hez, magát a külső programot fogjuk indexelni és nem az általa leírt dokumentumokat.
Miután a PDF-index elkészült, próbáljunk ki egy keresést:

```
% swish-e -f howto-pdf.index -m 2 -w boot
disk
```

Ilyesféle eredményt kell kapnunk:

```
1000 ../HOWTO-pdfs/Bootdisk-HOWTO.pdf
       "Bootdisk-HOWTO.pdf" 127194
983 ../HOWTO-pdfs/Large-Disk-HOWTO.pdf
      "Large-Disk-HOWTO.pdf" 85280
```

A `MetaNames` utasítás alkalmazása folytán címek és a PDF-fájlok útvonala szerint is kereshetünk:

```
% swish-e -f howto-pdf.index -w
swishtitle=apache
% swish-e -f howto-pdf.index -w
swishdocpath=linux
```

A keresések minden kombinációja támogatott, például:

```
% swish-e -f howto-pdf.index -w ´(larry and
wall)
    OR (swishdocpath=linux OR
swishtitle=kernel)´
```

Az előbbi példában azért szükséges idézőjeleket használnunk, hogy a zárójeleket megóvjuk a héjprogram értelmezésétől.

Súgóoldalak indexelése

Utolsó példánkban megmutatjuk, hogyan készíthetünk súgóoldalainkhoz hasznos és hatékony indexeket, és miként használhatjuk a `SWISH::API` Perl-modult az index-keresőfelület kialakításához. Ahogy eddig is, a munkát a beállításfájl elkészítésével kezdjük:

```
# sman-index.conf
IndexFile ./sman.index
  # elkészítendő index.
IndexDir  ./sman-index-prog.pl
IndexComments no
  # a megjegyzések szövegeit nem indexeljük
UseStemming yes
MetaNames      swishtitle desc sec
PropertyNames              desc sec
```

A legtöbb utasítás jelentését már korábban bemutattuk, most azonban néhány új `MetaNames` nevet is megadtunk, illetve egy új, `PropertyNames` nevű parancsot vezettünk be. Dióhéjban összefoglalva: a `MetaNames` határozza meg, hogy mi alapján keressen a SWISH-E. Az alapértelmezett `MetaName` a `swishdefault`, azaz ha a lekérdezésben nem adunk meg `MetaName` kapcsolót, akkor ez alapján fog keresni. A `PropertyNames` a visszaadott találatok leírására használható mező.
A SWISH-E által visszaadott eredmény általában az `Auto Properties` (önműködő tulajdonságok parancs) szerint jelenik meg, ilyen a `swishtitle`, `swishdesc`, `swishrank` és a `swishdocpath`. A beállításunkban olvasható `MetaNames` utasítás azt jelenti, hogy egymástól függetlenül nemcsak a teljes dokumentum, hanem csak a címek, a leírás vagy a szakasz alapján is keresni szeretnénk. A `PropertyNames` sor mutatja meg, hogy minden egyes találat visszaadásakor a `sec` és `desc` tulajdonságot szeretnénk látni, vagyis az oldal szakaszát (`sec`) és rövid ismertetőjét (`desc`).
A súgóoldalak XML formátumúvá alakítását és SWISH-E fejlécekké formázását az *1. listában* látható módon végezzük (***sman-index-prog.pl***).
Az 1. lista első ciklusa lesz a program fő váza. Itt pillantunk bele minden egyes súgóoldalba, szükség szerint értelmezzük, XML-é alakítjuk át, majd a SWISH-E igényeinek megfelelő fejlécekkel egészítjük ki:

- A `get_man_file()` a `File::Find` függvényt használja a súgóoldalak könyvtáraiban, hogy megtalálja a feldolgozandó súgófájlokat.
- A `make_xml()` és az `escape()` együtt a `parse_man()` által visszaadott `hashref`-ből készíti el az XML formátumot.
- A `parse_man()` végzi el a trükkös részt, kiemelve a szükséges mezőket a súgóoldal forrásából.

Most, hogy már megismertük a működését, használjuk is aprogramunkat:

```
% swish-e -c sman-index.conf -S prog
```

Miután ezzel megvagyunk, a `swish-e -w` kapcsolóját alkalmazva a korábbiakhoz hasonló módon kipróbálhatjuk a rendszert.

Végül nézzük meg, hogyan használhatunk `SWISH::API`-t alkalmazó Perl-parancsfájlokat az imént elkészített indexhez, létrehozva ezzel a unixos `apropos` parancs egy fejlettebb változatát. A kódot a 2. listában (51. CD Magazin/SWISH-E könyvtárában) találjuk (sman). Lássuk röviden a szerkezetet: az 1–14. sor az alapértékeket állítja be és a parancssori kapcsolókat értelmezi, a 15–23. sor kezeli a lekérdezést, illetve felületes hibakezelést végez, végül a 24–39. sor jeleníti meg a lekérdezés eredményét a `SWISH::API`-n keresztül visszakapott `Properties` alapján.
A Perl-ügyfél ilyen egyszerű. Próbáljunk meg súgóoldalainkból kikeresni egy témát:

```
% ./sman -m 1 boot disk
```

Az alábbit kell visszakapnunk:

```
ootparam (7) Introduction to boot time para...
```

Ugyanakkor már a következőképpen is kereshetünk:

```
% ./sman sec=3 perl
```

Ezáltal a keresést a 3. szakaszra korlátozzuk. Továbbá az `sman` program elfogadja a `--max=#` parancssori kapcsolót, ami korlátozza a visszaadott találatok számát, a `--file` kapcsolót, ami megmutatja a keresett szavakat tartalmazó fájl nevét, és a `--rank` kapcsolót, ami az adott lekérdezésen belül kapott rangot mutatja meg:

```
% ./sman --max=1 --file --rank boot
```

A fenti parancs a következő eredményt adja:

```
1000 lilo.conf (5) configuration file for lilo
     /usr/man/man5/lilo.conf.5
```

Figyeljük meg, hogy a rang az első, míg a forrásfájl az utolsó oszlopba kerül.
A sman csomag fejlettebb változatát a
➲ http://joshr.com/src/sman/ címen érhetjük el.

Összefoglalás

A SWISH-E rendszerének van két említésre érdemes árnyoldala is. Először is: csak a 8 bites ASCII adatokat kezeli. Másodszor: a SWISH-E indexből nem lehet elemeket törölni, a törléshez a teljes indexet újra létre kell hozni. A mérleg másik serpenyőjében a SWISH-E számos olyan képességét találjuk, amit itt még csak megemlíteni sem tudtunk. A részleteket a SWISH-E honlapján, a ➲ http://www.swish-e.org címen találhatjuk meg.

A cikkhez tartozó listák megtalálhatóak az 51. CD Magazin/SWISH-E könyvtárában.

Linux Journal 2003. július, 111. szám

Josh Rabinowitz
13 éve a programipar veteránja, aki tudását a NASA Ames-i kutatóközpontjában, illetve a CNET.com-nál és egyéb webcégeknél csiszolgatta. Jelenleg független tanácsadó és a SkateboardDirectory.com kiadója.

Inverz kinematika a Blenderben

Ismét eljött az ideje, hogy új tudásra tegyünk szert a Blender képességeinek megismerése során.

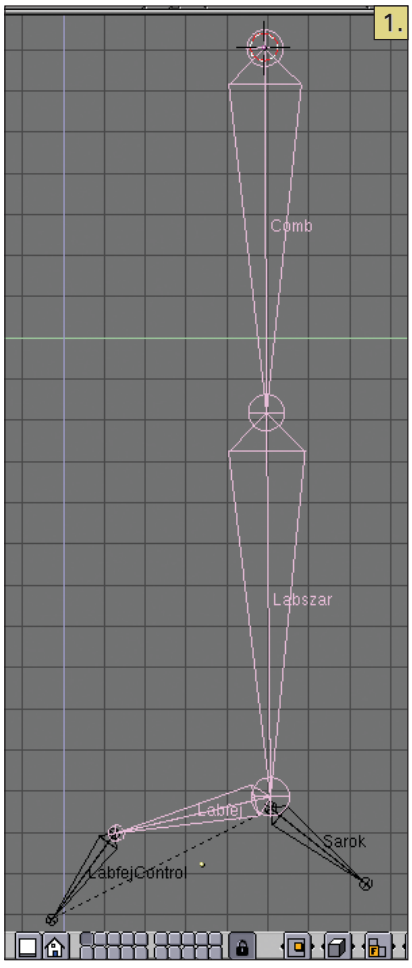
Remélem, kedves olvasóim, mára már sikerült kellőképpen elmélyedni a korábban elsajátított ismeretekben, így a továbbiakban nem fog gondot okozni a program használata és különféle lehetőségeinek megértése.
Ebben a hónapban azt szeretném röviden bemutatni, hogy karaktereinkhez hogyan készíthetünk inverz kinematika által mozgatott csontvázakat. Kezdetben egy egyszerű módszert szeretnék ajánlani a karaktermodellezés mikéntjével kapcsolatban. Egy csomó felesleges munkától kímélhetjük meg magunkat, ha betartjuk az alábbi sorrendet.

A terv

Először gondoljuk át, hogyan fog kinézni a karakter, majd készítsünk minél részletesebb rajzokat róla. Ezeket később a modellezés során is felhasználhatjuk, hiszen a pontos modellezéshez elengedhetetlenek. Természetesen a mintázatokat is könnyebben elkészíthetjük, ha egyszer már megrajzoltuk őket. Készítsünk elöl-, oldal- és felülnézeti képeket, és a modellezőprogramban használjuk őket háttérképként és segítségként egyaránt. A rajzok mellett legalább elképzelés szintjén szükségesnek tartom a háromszögek elhelyezkedésének átgondolását is. A következő lépésben tervezzük meg, hogy milyen mozgásokat kell végeznie a figurának, és gondoljuk át a csontváz szerkezetét. Modellezés során a csontváz létrehozásával kezdjük, erre könnyebben felépíthetjük majd a figurát. Ezután következhet a modell elkészítése, majd a mintázatokat képolvasóval és rajzprogrammal elkészítjük és finomítjuk. Utolsó előtti lépésként a modell mintázatának elhelyezése és pontosítása következzen, ezek után a mozdulatsorokat modellezzük a csontváz mozgatásával.
A fentiek betartásával könnyebben és rövidebb idő alatt tudunk majd mozgó figurákat modellezni, és szemet gyönyörködtető animációk vagy játékok készítéséhez használhatjuk fel őket.
Most tehát ismerkedjünk meg az inverz kinematika létrehozásának lehetőségeivel!

Mozgás!

Mint azt a korábbi részekben említettem, az inverz kinematika teszi lehetővé, hogy amikor valamilyen tárgyat mozgatunk, annak mozgása ne csak önmagában létező legyen, hanem a mozgás során az összekapcsolt tárgyak egymásra hatása is megjelenjen. Egyszerű példaként az emberi kar mozgása hozható fel, hiszen nem helyezhetem tetszőleges helyre a kézfejemet anélkül, hogy az alkar és a felkar is ne mozogna, és a kézfejhez kapcsolódó testrészek hosszúsága is korlátozza a kézfej mozgását.
Úgy gondolom, ennyi bevezető után elkezdhetjük a gyakorlati munkát. Egy három ízből álló csontváz mozgását tanuljuk meg, ennek alapján a későbbiekben bonyolultabb vázakat is képesek leszünk mozgásra bírni. Oldalnézetben hozzunk létre egy három részből álló vázat a ***Főmenü/Add/Armature*** menüpontok kiválasztásával. Az egyes részeket nevezzük el fentről lefelé, a „Comb”, „Lábszár” és a „Lábfej” megnevezéseket használva. Ehhez ki kell jelölni a vázat, majd a TAB billentyű alkalmazása után minden részt is az A billentyűvel. Ekkor az F9 billentyűvel váltsunk át a szerkesztőnézetbe, majd a nézet közepe felé látható „Bone” neveket változtassuk a fentieknek megfelelőkre. Ezt csupán azért kellett megtenni, hogy a kedves olvasók hozzászokjanak a nevek használatához, hiszen amikor emberi csontvázat készítünk, könnyebben eligazodunk az ismerős nevek között, mint a számozott csontok között. Természetesen a továbbiak megértése is könnyebben megy majd, ha például a leírás szerinti „Comb” megtalálásához nem kell minden esetben a szerkesztőnézetben keresgélni a csontok között. Miután elneveztük a csontokat, még egy két részből álló vázat kell létrehozni, amivel majd befolyásolhatjuk a mozgást. A szerkesztőmódban jelöljük ki a boka helyén lévő kör alakú csatlakozási pontot, majd a SHIFT-S billentyű hatására megjelenő menüből válasszuk a legalsó pontot. Így a térbeli kurzor pontosan a boka helyére kerül. Szüntessük meg a kijelölést, és az előbbi módon hozzunk létre egy új vázat. Mivel ennek az irányító váznak a részei nem csatlakoznak egymáshoz, újra szüntessük meg a kijelölést, és a fő vázon a lábfej végét kijelölve helyezzük el ide a kurzort (a SHIFT-S ismételt alkalmazásával). Szüntessük meg ismét a kijelölést, majd a vezérlő vázat kijelölve váltsunk szerkesztőmódba. Itt következik a másik vezérlőrész létrehozása, vagyis a főmenüből ismételten válasszuk ki az ***Armature*** pontot. Ezek után a létrehozott csontoknak adjuk a „Sarok”, illetve a „LábfejControl” nevet. Amennyiben mindent jól csináltunk, a továbbiakban



1. kép Kiindulás az inverz kinematikához

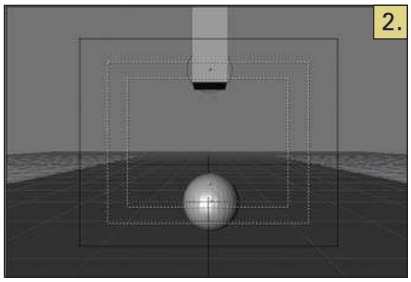
az *első* képen látható szerkezettel dolgozhatunk. Megjegyzem, hogy a képen a mozgatandó váz van kijelölve, tehát az látható rózsaszínnel jelölve, a fenti lépések után azonban a Blenderben még nem látható a „LábfejControl”-t és a „Sarok” vastagabbik végét összekötő szaggatott vonal. Ez a vonal jelenti azt, hogy a két csont alárendelt viszonyban áll egymással, és a helyes mozgáshoz ezt is be kell állítani. Ezt a két csontot a szerkesztőmódban kijelölve a szerkesztő nézetben megjelenik a csontok neve és az is, hogy milyen viszonyban vannak egymással. Ezt az információt a név mellett látható ***child of*** mező tartalmazza, és nekünk kell most beállítani a „Sarok”-nak ezt a tulajdonságát. Tehát a „Sarok” sorában válasszuk ki a legördülő listából a „LábfejControl” csontot. Így a létrehozott szerkezet már valóban megfelel az első képen láthatónak.

Inverz kinematikai rendszer

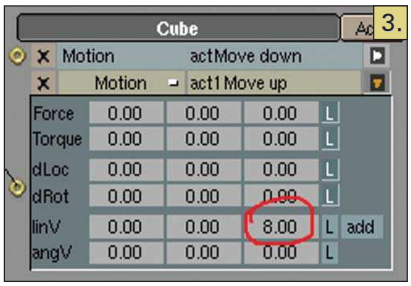
Ezek után nem sok teendőnk van, csupán a Blenderrel kell tudatni, hogy ezek a részek tulajdonképpen egy inverz kinematikai rendszert alkotnak. Itt lesz szükségünk az előző részben olvasottakra, nevezetesen a ***helyzetmód*** alkalmazására. Először azonban az esetleges elfordulásokat és az egyéb változtatásokat mindkét vázrészen meg kell szüntetni. Ezt az Alt-R billentyűkkel érhetjük el, de ne lepődjünk meg, ha a vázak helyzete megváltozik, hiszen így alaphelyzetbe kerülnek, ami megkönnyíti további munkánkat. Végezzük el ezt a műveletet mindkét vázon. Ezután jelöljük ki a három részből álló csontvázat, majd a Ctrl-Tab billentyűkkel váltsunk ***helyzetmódba***. Néhány ablakkezelő ezt a billentyűkombinációt az alkalmazások közötti váltásra tartja fenn, ezért helyette használhatjuk a nézet alján lévő gombok közül azt, amelyiken egy sárga fejecske látható. Ez a rétegek megjelenítéséért felelős gombok után a tizenkettedik.
Amikor ebben a módban dolgozunk, a kijelölt csontváz kék színnel jelenik meg. A sikeres váltás után jelöljük ki a „Lábfej”-et, és a szerkesztőnézet bekapcsolása mellett látható, láncszemeket ábrázoló gomb segítségével váltsunk át a korlátozásokat és kapcsolatokat meghatározó nézetre. Itt kell majd megadnunk, hogy melyik rész milyen kapcsolatban van a többiekkel. Az ***Add*** gombbal hozzunk létre egy új elemet, és a megjelenő vezérlők között keressük meg a ***Constraint type*** mezőt. Ez a kis x mellett található, és a listából ki kell

Alt-R	Elfordulás törlése
F8	Váltás játéknézetre
P	Játék indítása
Esc	Játék leállítása

választanunk az `IK Solver` típust. Ennek hatására a típus alatt megjelenik egy `OB` szerkesztőmező, ahová be kell írni azt az objektumot, amellyel vezérelni fogjuk a csontváz mozgását. Ebben az esetben nem neveztük át a két csontból álló vázat, így annak neve ***Armature.001*** maradt. Ezt kell a billentyűzet segítségével beírnunk az `OB` mezőbe. Ahogyan ezzel elkészültünk, megjelenik egy újabb szerkesztőmező, ahová a „Sarok” nevet kell beírni, hiszen ez a csont fogja vezérelni a teljes csontváz mozgását. Ezzel készen is vagyunk. Szüntessük meg a kijelölést, és a két részből álló váz mozgatásával máris egy működő rendszert láthatunk.
Felmerülhet a kérdés, hogy ha ez ilyen egyszerű, akkor miért volt szükség a „LábfejControl” vázrész létrehozására. Nos, ezzel tudjuk majd a lábfej elfordulását szabályozni. Ennek eléréséhez azonban újra az előbbi korlátozó tényezőkhöz kell újakat adnunk. Jelöljük ki megint a fő vázat, majd helyzetmódba váltva a „Lábfej”-et is. Adjunk újabb korlátozó elemet az ***Add*** gombbal a vázrészhez, ezúttal azonban az alapértelmezett ***Track to*** típusú elemen a típust hagyjuk változatlanul. Az `OB` ismét az ***Armature.001*** legyen, azonban az elemhez tartozó csontként most a „LábfejContol”-t kell beállítani. Az eredményeket megszemlélve látható, hogy a vázrendszer a „Sarok” mozgatásával most megfelelően mozgatható, és a két részből álló váz forgatásával a lábfej külön is képes forogni. Ha jobban megfigyeljük, akkor azt is észrevehetjük, hogy a mozgás nem tökéletes. Néhány helyzetben az egész láb kifordul, nem tudjuk például vízszintesen előrenyújtani. Ezen további korlátozó elemek létrehozásával könnyen segíthetünk. Szintén a fő vázon kell dolgoznunk, de első lépésben a „Comb”-ot jelöljük ki. Adjunk egy újabb ***Track to*** elemet a vázrészhez, ennek célját állítsuk az ***Armature.001*** objektum „LábfejControl” csontjára. A következő lépés legyen a „Lábszár” elemhez egy újabb korlátozó tényező hozzáadása, célja szintén az ***Armature.001*** váz, de itt a követendő csont a „Sarok” lesz. Ezeket a lépéseket megtéve már minden pozícióban helyesen fog állni



2. kép Kiindulás a játékhoz

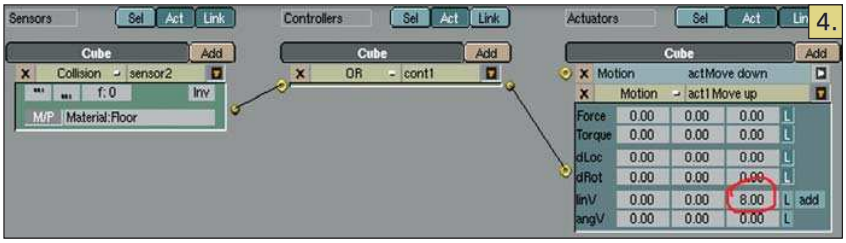


3. kép A kezdősebesség beállítása

a háromízű váz. Itt kell megemlítenem, hogy a ***Track to*** elemet más esetekben is eredményesen használhatjuk, például amikor a kamerának követnie kell egy objektum mozgását – alkalmazzuk bátran ezt az elemet. A ***Copy Location*** és a ***Copy Rotation*** elemek használata magától értetődő, így ezekkel most nem foglalkozom részletesen.

Játék megalkotása

Úgy gondolom, hogy mostanra már eleget tudunk a Blenderről ahhoz, hogy elkészítsük első egyszerű játékunkat. A jobb érthetőségért célszerű tisztáznunk néhány alapvető dolgot.
A Blenderben a játékmotorban négyféle elemtípust találhatunk. Az egyik a tulajdonság, amely egy-egy tárgyunk sajátja, és típusa a különféle programozási nyelvekben megszokottaknak megfelelően lehet egész, valós, logikai és szöveg. Itt lehetőségünk nyílik egy úgynevezett ***Timer*** típusú tulajdonság használatára is, ennek kezdőértéket adhatunk, majd az értéke önműködően növekszik. A második típus az érzékelő, amellyel különféle események bekövetkezésekor indíthatunk el valamilyen folyamatot. Ennek típusa is többféle lehet, itt azonban csak a ***Keyboard***, a ***Always*** és a ***Collision*** típusú érzékelőket fogjuk használni – ezekkel részletesen foglalkozom. A ***Keyboard*** alkalmas különféle billentyűzettől érkező események figyelésére, megadható, hogy melyik billentyű lenyomására legyen érzékeny, milyen módosítóbillentyűkkel



4. kép A pattogás logikája

együtt figyelje a billentyű lenyomását, és hogy amikor az adott billentyűt lenyomtuk, akkor a tárgyunk milyen tulajdonsága változzon meg. Az ***Always*** típusa érzékelő, tehát a megadott gyakorisággal jelzéseket továbbít a kimenetén. A ***Collision*** alkalmas arra, hogy egy megadott tulajdonsággal vagy anyagtípussal rendelkező tárggyal való ütközést érzékeljen.

Az eddigieket próbáljuk megvalósítani a gyakorlatban is. Kiindulásként hozzunk létre egy gömböt, egy síklapot és egy kockát, majd ezeket helyezzük el a *2. képen* látható elrendezésben.

A játék célja az lesz, hogy egy akadálypályán keresztül kijussunk a gömbbel a kijáraton. A kijárat a pálya másik végén lesz.

Kezdetben adjunk anyagot a síklapnak, és ezt az anyagot nevezzük „Floor”-nak. Ezután a kockát kijelölve az F8 billentyűvel kapcsoljunk játéknézetbe. Itt állíthatjuk be a játék működését meghatározó tulajdonságokat, érzékelőket és egyéb, a működést befolyásoló elemeket, amelyekről a későbbiekben bővebben szólok. Most a kocka kijelölése után kapcsoljuk be a nézet bal felső sarkánál található `Actor` kapcsolót, ezzel tudatjuk a Blenderrel, hogy ennek a tárgynak a tulajdonságait a játék futtatása során figyelembe kell vennie. Kapcsoljuk még be a megjelenő `Dynamic` kapcsolót is. Ezzel a fizikai törvények is hatni fognak a tárgyra, ami most elsősorban a gravitáció miatt lesz fontos, hiszen ez lesz majd az egyik akadály, a labdának ami a pálya síkja felett pattogni fog. Ha most a P billentyűvel elindítjuk a játékot, akkor már láthatjuk is a mozgást, vagyis azt, hogy a tárgy elkezd lefelé esni, és amikor érintkezik a síklappal, akkor megáll. Ez már fél siker, célunk ugyanis a folyamatos pattogás. Ezt úgy tudjuk megvalósítani, hogy egy érzékelőt helyezünk a tárgyra, ami a síklap elérésekor jelez, és ekkor a kockának új irányt adunk.

Tehát a következő lépésben a nézet közepe felé a ***Sensors*** felirat alatt találunk egy gombot, amivel új érzékelőt adhatunk a tárgyhoz. Az ***Add*** gomb alkalmazásával hozzunk létre egy új elemet, ennek típusát pedig a megjelenő legördülő listából kiválasztva állítsuk ***Collision***-ra. Itt található még egy ***M/P*** kapcsoló is, amivel választhatunk, hogy az adott anyaggal vagy tulajdonsággal rendelkező tárggyal való ütközéskor jelezzen-e meg az érzékelő. Ezt a kapcsolót is kapcsoljuk be, majd a mellette lévő mezőbe írjuk be a „Floor” szócskát. Ezzel elértük, hogy az érzékelő olyan tárgyakkal való ütközéskor jelezzen, amiknek az anyaga „Floor” nevű. Ilyen tárgy jelenleg egy szerepel a jelenetben, a síklap. Az érzékelő beállítása után már csak azt kell megadnuk, hogy mi történjen a jelzés hatására. A nézet jobb oldalán találhatók az események eredményeként létrejövő hatások. Itt is alkalmazzuk az ***Add***-ot, és az alapértelmezett hatás megjelenése után állítsuk be annak jellemzőuit. Amikor a kocka eléri a síklapot, arra van szükség, hogy újra felfelé induljon el. Úgy biztosítunk számára kezdősebességet, hogy a `linV` érték Z irányú összetevőjének pozitív értéket adunk meg. Ez látható a *3. képen*, a szükséges változtatás pirossal van jelölve. Én a példámban 8-as kezdősebességet határoztam meg, ami arra elegendő, hogy a kocka addig a magasságig pattanjon fel, ahonnan elindult.

Rugalmasság

Ideje, hogy egy, az eddigiek során kissé elhanyagolt anyagtulajdonsággal is foglalkozzam, a dinamikai tulajdonságokkal. Az anyagszerkesztőben a színek meghatározása mellett találhatunk egy ***Dyn*** feliratú gombot, ezt bekapcsolva megadhatjuk az adott anyag rugalmasságát, tapadási együtthatóját és azt, hogy mennyi perdületet szerezzen az ütközés után. Ezek használatához azonban a játéknézetben is be kell állítani a ***Actor*** gomb alatt található `Do Fh` kapcsolót.

Miután létrehoztuk az érzékelőket, és meghatároztuk, hogy a jelzés hatására a tárgynak milyen tevékenységet kell végeznie, már csak a kettő közötti kapcsolatot kell meghatározni. Erre szolgálnak az érzékelők és a hatások közötti területen lévő vezérlők. Ezeknek négy fajtája létezik, az `AND` ***és*** kapcsolatot jelent a kapcsolódó bemeneti jelzések között, az `OR` ***vagy*** kapcsolatot, míg a fennmaradó kettő működése általam jelenleg nem ismert. Szerencsére ez nem jelent gondot, hiszen a meglévő érzékelők és kapcsolatok elegendőek a játék elkészítéséhez. Válasszuk ki tehát a ***vagy*** kapcsolatot, és kössük össze a kimeneteket és bemeneteket. Ezek a sárga színű körök és a telt körök az egyes elemek mellett. Telt körrel jelölve láthatjuk a kimeneteket, amelyeket egyszerű kattintással és húzással az üres körrel jelölt bemenethez köthetünk. Végső lépésként kössük az értékelő kimenetét egy ***vagy*** kapcsolat bemenetéhez, majd ennek kimenetét a mozgatást megvalósító hatás bemenetéhez. A *4. képen* látható a helyesen kialakított vezérlőrendszer.

Ebben a részben eddig jutottunk, az összefoglaló táblázatban mindenki megtalálja a tanult billentyűkombinációkat; a következő hónapban pedig a billentyűzetkezeléshez használható érzékelőket mutatom be majd, és folytatjuk a játék elkészítését.

Fábián Zoltán
(dzooli@freemail.hu)
Programozóként dolgozik. Szabadidejében szívesen kirándul, és szeret rajzolni, érdekli a 3D grafika.

A PHP5 újdonságai

Újraírt Zend-motor, gyorsabb futás és továbbfejlesztett objektumkezelés – mindez év vége táján várható.

Nyár van, mégsincs uborkaszezon a PHP háza táján. Június végén a nagyközönség elé tárták a PHP 5.0.0-s próbaváltozatát. A fejlesztés tehát eljutott első nyilvános megjelenéséig (eddig csak CVS segítségével lehetett beszerezni), s már látni lehet az alagút végét. Számos jelentős változás lesz a 4-es vonalhoz képest, de semmi olyan, ami miatt meglévő PHP-programjainkat újra kellene írnunk, legalábbis nem nagyobb mértékben, mint az a 3-asról a 4-esre váltás során szükségessé vált. Nagyszámú olyan újdonság jön viszont az 5-ös változattal, amit érdemes megismerni.

Névterek

Hasznos újdonság lesz a PHP életében a névterek bevezetése. Ez az apró kis adalék nagymértékben segítheti a nagyobb lélegzetű munkák szerkezetének a kialakítását, ráadásul külső osztálykönyvtárak és egyéb kódok anélkül könnyedén együtt használhatók, hogy bármiféle osztálynév-, állandó- (constans) vagy változónév-ütközés veszélye állna fenn. Márpedig ilyen jellegű galiba akár akkor is előfordulhat, ha csak egyszerűen többen dolgoznak ugyanannak a projektnek más-más részein, és ugyanazt a változónevet eltérő célból veszik használatba.
A névterekkel némi logikai csoportosításra is lehetőségünk nyílik, az összetartozó osztályokat, állandókat egy térbe vonhatjuk. Névtereket létrehozni a `namespace` kulcsszó segítségével tudunk. Lássunk egy egyszerű példát (lásd az *1. listát*)!
A példakód természetesen csak szemléltetésre alkalmas, de a lényeg látható. A namespace „kulcsszót” egy név követi, amellyel a továbbiakban hivatkozni tudunk rá. A terület körülhatárolására hagyományosan a kapcsos zárójelek hivatottak. A névtér egy osztályhoz vagy függvényhez hasonlóan saját hatókörrel rendelkezik. Ez annyit tesz, hogy minden, a névtérben létrehozott függvény, változó, állandó a névtéren belül hozzáférhető, míg azon kívül közvetlenül nem. Viszont elérhetők, ha egy kis előtaggal fejeljük meg, azaz a névtér nevével és egy kettőzött kettősponttal. A névtér előtagot és az utána következő változó-, állandó-, függvény-, illetve osztálynevet hívjuk az objektumközpontú világban „minősített névnek”. Ilyen formában nemcsak a névtereken kívül tudunk hozzáférni az azokon belül létrehozott elemekhez, de ilyen módon az egyik névtérből át is nézhetünk a másikba.
A PHP-értelmezőnek az sem okoz gondot , ha névterünket esetlegesen több darabban hozzuk létre, azaz ugyanazt a névteret akár több állományban is megnyithatjuk, hogy elemeket pakoljunk bele. A névterek egymásba is ágyazhatók, azaz névtereken belül is létrehozhatunk külön névtereket. A kívülről való hivatkozáshoz a minősített névben a szülőnévtértől kezdve a teljes névtércsaládfát le kell írnunk. A névtérhivatkozásokat azonban egymástól csak egyszeres kettősponttal szükséges elválasztanunk (lásd az 1. listát).

1. lista A namespace.txt

```
?php

namespace Mail {
     const FELADO = 'kosar@hajo.hu';
     $level_darab = 0;
     function jelentes($cimzett, $szoveg) {
          mail($cimzett, 'Jelentes', $szoveg,
               ➥'From: '.FELADO);
     }
}

echo Mail::FELADO;
echo Mail::$level_darab;
Mail::jelentes('kapitany@hajo.hu', 'Szárazföld!');

?>
```

3. lista Az object-overloading.txt

```
<?php

class Gyenge_pelda_tulterhelesre {

     __call($fuggveny, $parameterek) {
          if ($fuggveny = 'tukroz') {
               if
(is_numeric($parameterek[0])) {
                    return $this->
                    ➥tukroz_szamot
                    ➥($parameterek[0]); } else {
                    return $this->
                    ➥tukroz_szoveget
                    ➥($parameterek[0]);
               }

          }
     }

     private function tukroz_szamot($szam) {
          return 0 - $szam;
     }

     private function
tukroz_szoveget($szoveg) {
          return sttrev($szoveg);
     }

}

?>
```

Egyelőre még semmi sem biztos...

A PHP eljövendő új változatáról még nem sok biztosat lehet elmondani, bármi és bármikor megváltozhat a végleges kiadásig. Néhol igen hosszúra nyúlt viták következményeképpen a dolgok megerősödnek, eltűnnek vagy megváltoznak. A cikk írása közben például kiderült, hogy egyre kevésbé valószínű, hogy a névterek bekerülnek a végleges ötös változatba. Még ez a játszma sincs lejátszva, sokan kardoskodnak a dolog mellett, de vannak erős érvek az elhagyása mellett is. Jelenleg az utóbbi áll nyerésre. A nehézség a nem teljesen tökéletes megvalósíthatóságban rejlik. Az egyik ellenérv szerint: írásmódját tekintve a „(feltétel) ? ha_igaz : ha_hamis" megoldással is ütközik a névtereket egymástól elválasztó kettőspont használata. Ez a kódba történő kisebb belenyúlással a megfelelő helyeken kiküszöbölhető. De természetesen nem csak ez a gond vele. Csekély sebességcsökkenés is jár a dologgal, és a sok hátrány mellett többek szemében egyre inkább eltörpülnek a névterek által nyújtott előnyök. Mindenesetre most sem tartják valószínűtlennek, hogy a névtértámogatás bekerüljön az új kiadásba, de ehhez egyelőre hiányzik a vállalkozó szellem: nincs aki választ tudna adni minden egyes felmerülő nehézségre. Reménykedjünk… Egy másik változás a MySQL-támogatásban lesz. A PHP és a MySQL együttműködésének nagy hagyományai vannak. Egyes híresztelésekkel szemben az igazság az, hogy ez így is fog maradni. Mindössze annyi módosul, hogy a PHP a MySQL-támogatást illetően egy régebbi állapotba kerül vissza, azaz mindenképp külső *mysql* függvénykönyvtárra lesz szükség. Erre a MySQL és a PHP felhasználási szerződésbeli különbségei miatt került sor. Nagy nehézséget a dolog nem jelent. Egyedül annyi pluszmunkával fog járni, hogy fordításkor a `-with-mysql=<mysql_könyvtár_elérhetősége>` kapcsoló beállítására is szükség lesz.

Változások az objektumkezelésben

Az PHP 5-ös változatának megjelenésével nagy lépést tesz előre az objektumtámogatásban is. Jelenleg elég kevés és tökéletlen megvalósítás áll csak rendelkezésre, ami azért így is sok mindenre elég. A jelenlegi kódoknak az új értelmezővel is lehetőleg ugyanolyan jól kell futniuk, így az sem mindegy, hogy milyen módon változnak meg a dolgok.

Az egyik sokak által és régóta várt módosítás az objektum értékként való átadásának megvalósításában történt. A korábbi (jelenlegi) változatokban ugyanis a függvényértékként átadott objektumból egy teljes másolat készült, amely önálló életet élt, majd a függvénnyel együtt megszűnt létezni. Több okból sem jó ez így: egyrészt a függvény által kezelt objektum változásai nem hatnak vissza az eredetire (márpedig ez lenne az üdvözítő), valamint felesleges processzor- és memóriaterhelést is jelent ez egyben. Ez a gond eleddig is megoldható volt, ha magunk gondoskodtunk a referenciahivatkozás megfelelő helyekre történő biggyesztésével, de a nagy többségre elmondható, hogy fogalma sem volt a kérdésről (csak szép csöndben több memóriát evett a PHP, mint kellett volna).

A PHP ötös változata már másképp fog bánni az objektum típusú értékekkel, ugyanis önműködően referenciaként adja át őket. Hasonlóan fontos változás történik az objektumok létrehozó, illetve megszüntető (constructor/destructor) függvényeivel kapcsolatban. Létrehozó most is létezik a C++ nyelvben is ismert formában. Az osztály nevével azonos elnevezésű függvény hivatott az új objektum születésekor az előkészületeket megtenni, a változóknak alapértelmezett értéket adni. Viszont nem létezik destruktor, pedig az is hasznos eszköz lehet a tarsolyunkban. Mielőtt egy objektum megsemmisül, esetleg érdemes lehet még egy-két rutinfeladatot végrehajtatni, ilyen például a naplózás, esetleg a hibakeresésben segítő adatok kiíratása. Erre jelenleg nincs lehetőség. A C++-féle `~Osztálynév()` függvény létrehozása sem segít. A fenti feladatokat ugyan egy `register_shutdown_function()` függvényhívás segítségével is elvégezhetjük, de ekkor a záró műveletsor semmiféle közvetlen kapcsolatban nem lesz objektumunkkal.

Mit hoz a jövő? Egy új létrehozó-, illetve megszüntető-megadási lehetőséget. Az 5-ös változattól felfele a `__construct()` és a `__destruct()` tagfüggvények fognak gondoskodni az induláskori és a megsemmisülés pillanatában elvégzendő feladatok biztosításáról. Értelemszerűen a `__construct()` tagfüggvényben tudjuk megadni az éledéskori alapállapotot, a `__destruct()`-ban pedig az objektum megszűnésekori feladatokat tudjuk meghatározni.

Miért jobb ez az elnevezésmód? Tegyük fel, hogy egyre növekvő munkánkban meg szeretnénk változtatni az egyik osztályunk nevét. Ekkor hibalehetőséget jelenthet az, ha elfelejtjük a létrehozót is átnevezni, követve a névváltozást. Másrészt előfordulhat olyan eset is, hogy leszármaztatott osztályunk egyik tagfüggvényéből akarjuk meghívni a szülőobjektum létrehozóját. Ez a jelenlegi felállás szerint csak akkor oldható meg, ha a szülőobjektum neve ismert. Ez viszont nem feltétlenül egyértelmű. Az új elnevezés használatával viszont minden esetben meg tudjuk célozni a szülő létrehozóját. Természetesen nem kell megijedni, a PHP5 telepítése után nem kell a létrehozó függvényeket az összes osztályunk forrásaiban azonnal átnevezni. A visszafelé való csereszabatosság szem előtt tartásával a „régi", osztálynévvel megegyező létrehozó kerül (létezése esetén) végrehajtásra, ha a PHP nem lel a `__construct()` tagfüggvény nyomára.

Újdonságok az objektumkezelésben

- A `__clone()`
  Azzal a lépéssel, hogy az 5-ös változattól kezdve – ha tetszik, ha nem – másolat helyett hivatkozás adódik át a függvények számára értékként, elesünk attól a néha azért szükséges lehetőségtől, hogy mégiscsak egy teljes, az eredetivel azonos, de innentől önálló életet élő objektumot hozzunk létre. A feladat megoldására szolgál a minden objektummal együtt járó tartozék, a `__clone()` tagfüggvény. Példának okáért eztán függvényértéknek adott igény esetén nem a `$Dolly` objektumváltozót adjuk meg, hanem a `$Dolly->__clone()` által létrehozott másolatra való hivatkozást.
  No, azért ne elégedjünk meg ennyivel! A `__clone()` függvény az osztályban ugyanis egy saját, hasonnevű függvénynyel felülírható, ami átveszi a szerepét. Így teljes vezérlést nyerhetünk afelett, hogy mi és milyen módon kerüljön átadásra a klónozás során. A dolog hogyanja: a függvényen belül két objektumhivatkozás is rendelkezésre fog állni ehhez. Egyrészt a megszokott `$this`, amely már a klónozás termékére fog mutatni, míg a másolás forrását a `$that` objektumváltozóban kapjuk meg. Innentől ránk van bízva, mit és hogy teszünk át `$that`-ból `$this`-be, valamint miféle egyéb tevékenységeket folytatunk még eközben.
- A `__call()`
  Az új Zend-motor további újdonságokat is hoz. A `__call()` névre hallgató tagfüggvény segítségével minden olyan eljáráshívást elkaphatunk, ami mögött nem áll a névnek megfelelő tagfüggvény, esetleg csupán nem érhető el

4. lista A static.txt

```
<?php

class Math {
    static public $PI = 3.1415926536;
    static public abs($szam) {
        if ($szam >= 0) {
            return $szam;
        } else {
            return 0 - $szam;
        }
    }
}

$pi_erteke = Math::$PI;
$abszolut_ertek = Math::abs($valamennyi);

?>
```

5. lista Az abstract.txt

```
<?php

abstract class Alakzat {
    abstract function rajzol();
}

class Haromszog extends Alakzat {
    function rajzol() {
        // rajzolunk egy haromszoget
    }
}

class Kor extends Alakzat {
    function rajzol() {
        // rajzolunk egy kort
    }
}

?>
```

kívülről. Amennyiben létrehozunk egy ilyen `__call()` névre hallgató tagfüggvényt, az minden olyan esetben működésbe lép, ha a meghívott eljárás nem létezik vagy nem hozzáférhető. A függvénynek két értéket kell befogadnia: az elsőben érkezik a próbára tett eljárás neve, a másodikban pedig tömb formájában a függvényértékként megadott adatok futnak be. A dolog több mindenre is jó. Egyrészt így magunk határozhatjuk meg, hogy mi történjen olyankor, ha valaki nem létező függvényt próbál meghívni. Ennél viszont sokkal hasznosabb alkalmazási lehetősége is létezik ennek az eszköznek.

Ha nem is olyan egyszerű és önműködő formában, de létrehozhatjuk saját túlterhelt függvényeinket. Mit is jelent ez? A túlterhelt függvénynevek olyan nevek, amelyek mögött több megvalósítás is áll, és annak függvényében hívódik meg egyik vagy másik függvény, hogy épp milyen típusú és számosságú értéksort kapott. C++-ban, mivel az fordított és erősen típusos nyelv, a dolog viszonylag egyszerűen kivitelezhető. Ott egy adott függvényt nemcsak a neve, de a paraméterezhetőségének a miképtje is azonosít, így nem gond két különböző függvényt azonos néven létrehozni. A PHP viszont egy fordítás közben futó parancsnyelv, így a C++-féle módszer megvalósíthatatlan. A `__call()` függvényt mankónak használva azonban a dolog mégiscsak megoldható osztályokon belül, hiszen abból – az értékek típusának és számának megvizsgálása után – tetszőleges függvény meghívható. Szemléltetésül egy példa, ahol a hosszú nevű osztályunk `tukroz()` tagfüggvénye a valóságban nem létezik, azt két másik tagfüggvény helyettesíti: a `tukroz_szamot()` és a `tukroz_szoveget()` saját, belső (private) függvény (hogy jelen esetben mit jelent a belső szó, arról egy kicsit később lesz szó). A két függvény közül a megfelelő meghívásáról – a kapott érték megvizsgálása után – a `__call()` gondoskodik.

- A `__get()`, `__set()`

  Az objektumoknak a tagfüggvények mellett léteznek egyéb sajátjai is, például a saját változói. És léteznek bizony változók, értékek, amelyekkel az adott objektum nem rendelkezik. Az ilyenekhez való sikertelen hozzáférések esetére vethetjük be a `__get()` és a `__set()` függvényt. Az előbbi a nem létező változók olvasásakor, a másik pedig ezek írásakor lép működésbe, csakis akkor, ha létrehozunk ilyen függvényeket. A dolog tehát kísértetiesen hasonlít a `__call()` esetére. A `__set()` meghívásakor két értéket is fogadnunk kell. Az elsőben a nem talált vagy nem nyilvános változó nevét kapjuk meg, amibe adatot próbáltak kívülről adagolni. A másodikban pedig a beállítani kívánt értéket. Hogy mit kezdünk ezekkel az adatokkal, az már képzelőerő kérdése. A `__get()` esetén egyetlen fogadnivaló értékünk van: az olvasni próbált változó neve.

Private, Protected, Public

A fenti szavak sem csengenek ismeretlenül a más nyelvektől érkezők számára. Ilyen előtagokkal ruházhatunk fel minden osztályon belüli változó- és függvénymeghatározást. A legutóbbi, a `public` ezek közül a PHP számára eddig is követett módszert jelenti, azaz jelenleg minden osztályon belül létrehozott tagfüggvény és változó elérhető kívülről. A PHP5 használatba vételétől kezdve az osztályon belüli elemek hozzáférését is finomszabályozhatjuk. A `private` előtag használatával (ezt mind változómegadás, mind függvénymeghatározás elé odabiggyeszthetjük) levédhetünk bizonyos elemeket a külső meghívástól, olvasástól, illetve módosítástól. Az osztályon belüli tagfüggvények ettől függetlenül továbbra is hozzáférnek mindenhez. A belső (private) függvények használatára volt példa a többértelmű, túlterhelt tagfüggvény megvalósítását bemutató listában, ahol minden, az értékektől függő megvalósításért felelő függvény védett a külső, közvetlen meghívástól. A teljes külvilágtól való elzárás és a teljes nyilvánosság között akad még egy átmeneti lehetőség. Előfordulhat, hogy bár a közvetlen külső meghívás nem kívánatos, az adott osztályt kibővítő gyermekosztályokból erre mégiscsak szükség lehet. Az ilyen hozzáférési forma a védett, azaz `protected` típusú elérési mód. Ha ilyen előtagot biggyesztünk egy érték vagy függvény meghatározása elé, az az osztályból származtatott összes gyermekosztályból (de csakis onnan) ugyanolyan jól hozzáférhető lesz.

Static

Megeshet, hogy egy osztály olyan elemekkel is rendelkezik – legyen az függvény vagy éppen egy változó – amelyhez

6. lista Az exception.txt

```
<?php

try {
    if (!mail($cim, $tema, $szoveg)) {
        throw new Exception('A levél
elküldése sikertelen');
    }
} catch (Exception $e) {
    echo '<b>Sulyos kivétel:</b>' . $e-
>getMessage() . "\n";
    echo 'A hibás sor: ' . $e->getFile() .
', ' $e->getLine() . ". sor\n";
}

?>
```

anélkül hozzá lehet férni vagy esetlegesen meg lehet hívni, hogy az adott osztályt egy objektumban példányosítanánk. Ez olyankor lehetséges, amikor például egy adott függvény kimenetére az adott objektum pillanatnyi állapota egyáltalán nincs hatással. Szemléltetésül tekintsük meg a *4. listát*.

Objektumhivatkozások önműködő feloldása (dereferencing)

Objektumaink éppen más objektumokat is tartalmazhatnak, és függvényértékként adhatják vissza őket. Ilyenkor a jelen lehetőségek között egy objektumtól visszakapott objektum egy tagfüggvényét meghívni két lépésben megy csupán. A PHP5 ebben is hoz némi változást. Az eddig csak ilyenformán működő programsorok

```
$csap = $Leny->csap_noveszt();
$csap->kapaszkodik();
```

az ötös változattól kezdve ezzel a sorral lesznek helyettesíthetők:

```
$Leny->csap_noveszt()->kapaszkodik();
```

Absztrakt osztályok

A legtöbb osztály, amelyből további osztályokat származtatunk, önmagában is hasznos, példányosításra méltó típus. Létezik más nyelvekben, azonban a bázisosztálynak egy olyan fajtája, ami önmagában semmire sem jó, kizárólag más osztályok közös alapjaként szolgál, egyben megfogalmazva azok közös tulajdonságait. Az ilyen önállótlan bázisosztályokat absztrakt osztályoknak is szokás nevezni.
Egy osztályt meghatározásakor az `abstract` előtag használatával tudjuk megfosztani az önálló lét lehetőségétől. Az ilyen osztályok példányosítási kísérletei rendre kudarcot fognak vallani, csakis az ezekből származtatott alosztályok számára lehetséges az önálló lét. A teljes osztályon belül lehetnek helyben kifejtett tagfüggvények, de ezek külön is rendelkezhetnek az `abstract` előtaggal, ami annyit jelent, hogy az adott függvény kifejtése csak a származtatott osztályban fog megtörténni (de ott kötelező jelleggel, hiánya pedig hibaüzenethez vezet). Lássunk egy példát egy absztrakt osztályra és két bővítményére (*5. lista*)!

Közös felület

A PHP4-nek az a korlátja, hogy egyszeres öröklődésre nyílik csak lehetőség, a továbbiakban is fennmarad. Ez annyit jelent, hogy egy bázisosztályt kiegészítő gyermekosztály nem szolgálhat további gyermek bázisaként. Ily módon az absztrakciónak is megvannak a maga korlátai. A gond enyhítésére szolgál a közös felület (interface) létrehozásának a lehetősége. Egy ilyen felület egy olyan absztrakt osztálynak felel meg, amelynek minden tagfüggvénye is absztrakt formában (helyben nem kifejtve) létezik. A kulcsszavak, amikre a felületekkel kapcsolatban szükségünk van, az `interface` és az `implements`. Ezek rendre a hagyományos osztályok `class` és `extends` szavainak felelnek meg. A felületek nagy előnye, hogy a megvalósítás (implements) még nem számít származtatásnak, így a megvalósításra szánt osztályból még származtathatók gyermekek.

Több szülő közös gyermeke

Kellemes új lehetőség lesz az is, hogy egy származtatott osztályt több osztály közös gyermekeként is létrehozhatunk, vagy akár több felületet valósítunk meg egy osztályban.
A dolog trükkje mindössze annyi, hogy az `extends` és `implements` kulcsszavak után nem egy, hanem vesszővel elválasztva több bázisosztály/felület nevét adjuk meg.

Kivételkezelés

Általános nehézség az osztályokkal kapcsolatban, hogy a különféle osztálykönyvtárakat létrehozó programozók ugyan pontosan be tudnak határolni hibákat, de nem feltétlenül tudják, hogy mit kezdjenek vele. Ugyanígy az ilyen osztálykönyvtárakat felhasználó programozók már tudják, hogy melyik hibával mit kezdjenek, de nem tudják közvetlenül felderíteni őket. Sokszor szükség van rá tehát, hogy a hiba felderítésének és kezelésének helye (és szintje) ne okvetlenül legyen ugyanaz. Az ilyen hibák egységes kezelésére nagyszerű eszköz a kivételkezelés. És bizony, a kivételkezelés hármasa – a `try`, `throw`, `catch` csapat – is felüti a fejét a PHP következő nemzedékének kiadásában. Innentől pontosan meghatározhatjuk, hogy mit tartunk hibának, és hol és milyen módon óhajtjuk kezelni azt. Ráadásul azt az előnyét is élvezhetjük, hogy ilyen módon teljes mértékben elkülöníthető a „hasznos” kód és a hibák kezeléséért felelős kód.

A cikkhez tartozó listák megtalálhatóak az 51. CD Magazin/PHP könyvtárában.

Heiglig (Cece) Szabolcs (cece@phphost.hu)
Veszprémben él. Hobbija, kedvenc időtöltése és munkája is a programozás. Szabadidejében hajlamos kerékpárra pattanni, vagy baráti társaságban szerepjátékokkal foglalkozni.

KAPCSOLÓDÓ CÍMEK

A legfrissebb fejlesztői PHP5 a
➲ http://snaps.php.net címről gyűjthető be.
A Zend-motorról és változásairól több cikk és GYIK is szól
➲ http://zend.com
A PHP5 új fejlesztéseinek folyamatos és teljes leírását megkísérlő cikk a ➲ http://phpvolcano.com/articles/php5 címről tölthető le.

Programhonosítás a gettext csomaggal

Azt mondják, hogy a Microsoft annak idején azzal hódította meg a piacot, hogy a Windows 3.1-et a világ szinte minden nyelvén hozzáférhetővé tette. Ezáltal a számítástechnika széles rétegek számára lett elérhető.

A Linux terjedését gátló egyik fő ok régebben ugyanez volt. Nem várható el, hogy a gépet csak egyszerűen használni kívánó felhasználók az angol nyelvet is ismerjék. Sőt a számítástechnika szaknyelvében rengeteg olyan kifejezés akad, amit még az angolul egyébként jól tudóknak is külön meg kell tanulni. De a nemzetközi irodalmi művek is csak akkor válhattak egy nyelvi kultúra részévé, ha az adott nyelven is elérhetők lettek (gondoljunk csak a Biblia-fordításokra vagy Shakespeare műveire).

Igaz, az internet térhódításával néhány éven belül eljuthatunk oda, hogy az angol nyelvtudás olyan természetes lesz, mint a jogosítvány, mégis, egészen más érzés az anyanyelvünkön értő programmal dolgozni. Ezt a kereskedelmi programok készítői is pontosan tudják.

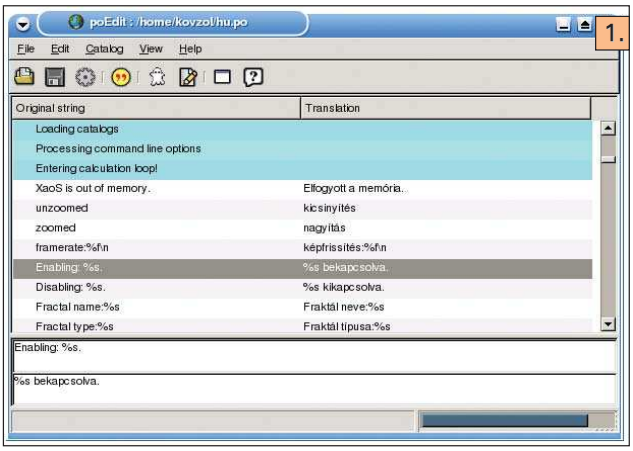
Az elmúlt években több nagy fajsúlyú program (KDE, Gnome, OpenOffice, Mozilla) magyarítása is nagy nyilvánosságot kapott. Bár Magyarország kis ország, és a honosítás előnyeit hasznosító felhasználók száma aránylag alacsony, e programok magyarra fordítása nagy előrelépés a Linux elterjedésében.

A honosítás olyan munka, amit – a programozói tudással ellentétben – csupán felhasználói ismeretekkel is el lehet kezdeni, és ez tovább növeli a Linux fejlesztőinek táborát. S mivel többen magukénak érzik a honosított programot, még többen vannak, akik a barátaikat is meg akarják győzni arról, hogy „ez a jobb, ezt használd!”. Izgalmas (s megmosolyogtató) látni azt a lelkesedést, amellyel a sokszor avatatlan újságírók a Linuxról írnak – önmagában az a tény, hogy az UHU Linux százszázalékosan magyar fejlesztés, olyasmi, amire egyszerűen büszkék lehetünk, és végre Magyarország is kihúzhatja magát, hogy van saját fejlesztésű operációs rendszere.

A lelkesedést a megfelelő mederben tartva valóban minőségi honosítások jöhetnek létre. A ➲ http://forditas.fsf.hu/html/Utmutato.html címen részletes nyelvi útmutatót olvashatunk ***Koblinger Egmont*** és ***Tímár András*** jóvoltából. Hasznosnak és fontosnak tartom a Linuxvilág Szótár rovatát is, amely szintén a nyelvi egységesítés felé mutat jó irányt.

Ha eldöntöttük, hogy csatlakozunk valamelyik program fordításához, akkor a legegyszerűbb felvenni a kapcsolatot a fejlesztői csapattal. Előfordulhat, hogy a programhoz még nincs magyar fordítás, de az is lehet, hogy már létezik, csak a fordító esetleg már nem foglalkozik a karbantartásával. Mindez az olvasó számára azt sejteti, hogy a fordító munkája akár hosszú évekig is elnyúlhat, amint a program új szövegekkel gyarapszik, esetleg bizonyos szövegrészek módosulnak benne.

Némelyik program fejlesztői erős iramot diktálnak komolyan betartandó határidőkkel (ami természetesen a fordítóra is vonatkozhat), más programoknál a lendület kisebb, és a fordítás akár évekig is elhúzódik. Azt mindenesetre tudnunk kell, hogy a programozást és a fordítást is többnyire önkéntesek végzik, szabadidőben; ez azonban nem ad felmentést az alól, hogy amennyiben egy fordítási munkát elvállalunk, azt



1.

2.

felelősséggel végezzük el. A nagyobb programoknál több fordítóra is szükség lehet a lefordítandó szövegek nagy száma miatt, így számítsunk arra, hogy csapatmunka várhat ránk. Bizonyos programok lefordításához komoly szakmai tudásra is szükség lehet. Készüljünk fel arra, hogy szinte minden programban akadnak olyan kifejezések, amelyeket csak akkor fogunk tudni lefordítani, ha a programot angol eredetijében már teljes egészében megismertük – csak ezután érdemes a munkának teljes erőbedobással nekivágni.

Technikai részletek

A programok nemzetköziesítésére több módszer is kínálkozik. Úgy tűnik, hogy a legéletképesebb és legelterjedtebb kezdeményezés a GNU gettext projekt, amely jelenleg a 0.12.1-es változatnál tart. A ➲ http://www.kde.org/fordfaq/forditas.html és a ➲ http://www.szabilinux.hu/forditasok/gnome-faq/i18n.html címeken elindulva jó alapokra tehetünk szert a témában, magyar nyelven, röviden. (A gettext hosszú, alapos leírása talán sokakat elriaszt ettől az egyébként nagyon kényelmesen kezelhető eszköztől.)

A gettext alapú fordítási folyamat lényege a következő: egy úgynevezett sablonfájlban (pl.: ***messages.pot***) a program szerzői az összes lefordítandó szöveget felsorolják angol nyelven. Egy ilyen szöveg lehet egy-egy menü neve, esetleg egy rövid felirat, vagy akár egy hosszabb, több mondaton is átnyúló magyarázó leírás. A sablonfájlt lemásolva a fordító létrehozza a majdani saját nyelvű fordításokat tartalmazó fájlt (például ***hu.po*** néven), egy-két perc alatt módosítja a fájl legelején található leíró fejlécet, majd az egyes angol szövegek alá begépeli a magyar nyelvű megfelelőt. Ezután a szerzőknek a ***hu.po*** fájlt elküldve a programba már könnyen befűzhetők a magyar nyelvű üzenetek (a befűzést a fejlesztők rendszerint maguk elvégzik). Nézzünk egy példát! A XaoS fraktálrajzoló program 3.1-es változatában a forrásprogram ***src/i18n*** könyvtárában található ***hu.po*** fájl így indul:

```
# XaoS NLS file for Hungarian language.
# Copyright (C) 2002 Free Software Foundation,
# Inc.
# Zoltan Kovacs <kovzol@math.u-szeged.hu>, 2002.
#
msgid ""
msgstr ""
"Project-Id-Version: XaoS 3.1\n"
"POT-Creation-Date: 2002-09-25 21:17+0200\n"
"PO-Revision-Date: 2002-08-17 21:44+0200\n"
"Last-Translator: Zoltan Kovacs
➥<kovzol@math.u-szeged.hu>\n"
"Language-Team: Hungarian\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain;
➥charset=ISO-8859-2\n"
"Content-Transfer-Encoding: 8-bit\n"

#: ../ui/ui.c:310
msgid "XaoS is out of memory."
msgstr "Elfogyott a memória."

#: ../ui/ui.c:360
msgid "%s %.2f times (%.1fE) %2.2f
➥frames/sec %c %i %i %i %i                "
msgstr ""
"%2$.2f-szeres %1$s (%3$.1fE) %4$2.2f kép/mp"
" %5$c %6$i %7$i %8$i %9H$i                "

#: ../ui/ui.c:361
msgid "unzoomed"
msgstr "kicsinyítés"

#: ../ui/ui.c:361
msgid "zoomed"
msgstr "nagyítás"

#: ../ui/ui.c:368
#, c-format
msgid "framerate:%f\n"
msgstr "képfrissítés:%f\n"
```

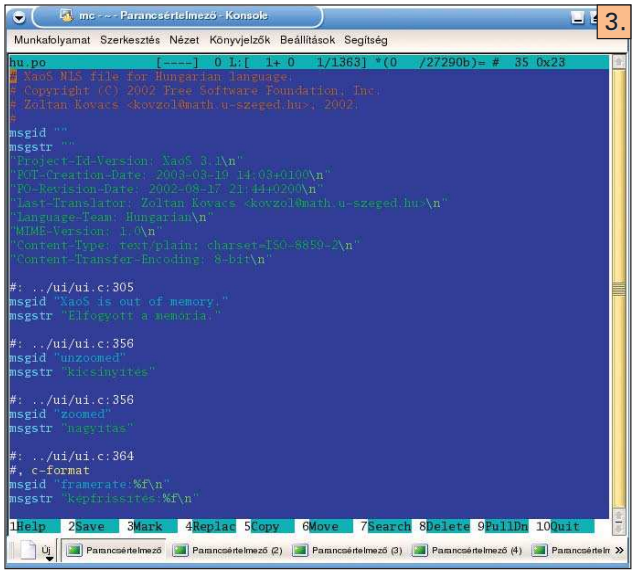
A # (kettős kereszttel) kezdődő sorok megjegyzések, tartalmuk másodlagos. Az angol nyelvű szövegeket az `msgid` kezdetű sorokban olvassuk, idézőjelek között. (Amennyiben a szöveg egy sornál hosszabb, úgy az a következő sorban is folytatható, ismételt idézőjelek között.) Az `msgstr` kezdetű sorok adják meg a szöveg magyar nyelvű megfelelőjét. Látható, hogy a ***.po*** fájl első „éles” bejegyzése az üres "" szöveghez kapcsolódik: a szabvány szerint az ehhez tartozó „fordítás” ad információt a ***hu.po*** fájl tartalmáról. (Ennek a formátuma eléggé kötött, és hibaüzenetet kapunk, ha valamelyik kötelező rovatát nem töltöttük ki helyesen.) Akik a C nyelvben kevésbé járatosak, azoknak könnyű a mind az angol szövegekben, mind a fordításukban megjelenő formázó részszövegeket felismerniük. Ilyen például a `\n`, ami az új sor jele, vagy a `%s` és a `%f`, amelyek arra utalnak, hogy a helyükbe egy másik szöveget vagy egy valós számot kell behelyettesíteni. Arra is mód nyílik, hogy a magyar változatban az egyes számokat, értékeket az angol eredetihez képest más sorrendben helyettesítsük be. Például az

```
msgid "Today is %s %s in the year %s."
msgstr "A mai dátum: %3$s. %2$s %1$s."
```

eredeti-fordítás párral elérhetjük, hogy az angol szöveg tükörfordítása helyett a magyar szokás szerinti megfelelő jelenjen meg a programunkban. Munkánk során azonban többnyire egyszerűbb esetekkel találkozunk. Az sem baj, ha kezdetben nem is fordítunk le minden szöveget, csak a legfontosabbakat: a kihagyott esetekben mindig az angol eredeti fog a magyar változatban is megjelenni. Látható az is, hogy a fenti szövegek többnyire nem szó szerint lettek lefordítva. Ennek oka sokszor az, hogy a magyar nyelv logikája néha egészen más, mint az angolé, s így kerülőutakra kényszerülünk.
A fordítási lépéssorozatot több grafikus segédprogram is megkönnyítheti. A régebben készült `ktranslator` program helyett újabban talán többen használják a poEdit többfelületes alkalmazást (ez ugyanis Windowson is elérhető), illetve a KDE alatt futó kbabel programot (*1–2. kép*). Akik mégis a „fapados” megoldásokat szeretik, azoknak viszont teljes szívvel ajánlható például a Midnight Commander szövegszerkesztője (*3. kép*).

A puding próbája

Sokáig – akár hónapokig is – eltarthat, amíg az elkészült ***hu.po*** fájlunk a fejlesztők jóvoltából belekerül az időszerű változatba, és élesben kipróbálhatjuk. Ha azonban ügyeskedünk egy kicsit, már a ***hu.po*** fájl készítése közben belepillanthatunk, hogyan is néz ki a honosított program magyarul, akár csak részben lefordított szövegekkel.



3.

4.

A ***hu.po*** fájlból a gettext csomag egyik segédprogramja egy ***.mo*** kiterjesztésű fájlt készít, amelyet fejlesztett programunk bináris változata könnyedén értelmezni tud. Ezt a fájlt (legyen most a neve ***ize.mo***, arra utalva, hogy az „izé” nevű programot fordítjuk magyarra) rendszerint a ***/usr/share/locale/hu_HU/LC_MESSAGES*** könyvtárban helyezik el (a Linux-változatunktól és a telepített programoktól függően ettől különböző, illetve ezenkívül más, hasonló nevű könyvtárak is szóba jöhetnek). Keressük meg a gépünkön ezt a könyvtárat, és nézzünk utána, hogy programjaink közül melyiknek van telepítve a magyar nyelvű változata.
A fenti segédprogram neve: `msgfmt`, futtatása a következő módon javasolt:

```
$ msgfmt -o /usr/share/locale/hu_HU/
➥LC_MESSAGES/ize.mo hu.po
```

Világos, hogy ehhez rendszergazdai jogosultság is szükséges. Ha ilyennel nem rendelkezünk, a saját könyvtárunkban egyszerűen hozzunk létre egy ***hu_HU/LC_MESSAGES*** alkönyvtárat, s oda dolgozzunk – ebben az esetben azonban lehet, hogy a fejlesztett programba is bele kell nyúlnunk, hogy tudassuk vele: a fordításokat nem az alapértelmezett könyvtárban kell keresnie.

Egy mintapélda

Elképzelhető, hogy egy teljesen új programot akarunk írni, amit nemzetközi támogatással szeretnénk ellátni. Programunk, az izé 1.0-s változatának ***ize.c*** nevű forráskódja a következőképpen néz ki:

```
#include <locale.h>
#define _(szoveg) gettext(szoveg)

main()
{
     setlocale(LC_MESSAGES, "");
#ifdef HELYBEN
     bindtextdomain("ize", getenv("HOME"));
#endif
     textdomain("ize");
     printf(_("Welcome to the program %s!"),
               ➥"ize");
     printf("\n");
}
```

Az első sorban tudatjuk a C-fordítóval, hogy a gettext csomaggal dolgozunk. A másodikban egy olyan rövidítést adunk meg, amellyel a későbbiekben elérhetjük, hogy minden szövegünk nemzetközi változata a `_("szöveg")` hívással is megjelenhessen (ez szabványos rövidítés, a gettext leírása is ezt a módszert javasolja, sőt a gettext segédprogramjai alapértelmezésben ezt a rövidítést támogatják). A `setlocale()` függvénnyel kapcsolunk nemzetközi üzemmódba (lehetőség volna arra is, hogy például a számokat, illetve a pénznemeket is magyar szabvány alapján jelenítsük meg, az `LC_NUMERIC` és `LC_MONETARY` változók beállításával; erről bővebben a gettext leírásában olvashatunk). Amennyiben nincs rendszergazdai jogosultságunk, az ***ize.mo*** fájlt a ***~/hu_HU/LC_MESSAGES*** könyvtárból olvassuk majd be, ezt engedélyezzük a `#ifdef` és a `#endif` közötti `bindtextdomain()` függvénnyel. A kiíratás a C nyelvben megszokott módon, a `printf()` függvénnyel történik.
Ezek után megadjuk a programunkban egyelőre egyetlen angol nyelvű szöveg fordítását a ***hu.po*** fájlban:

```
msgid ""
msgstr ""
"Project-Id-Version: ize 1.0\n"
"POT-Creation-Date: 2003-07-20 08:10+0200\n"
"PO-Revision-Date: 2003-07-20 08:32+0200\n"
"Last-Translator: Zoltan Kovacs
<kovzol@math.u-szeged.hu>\n"
"Language-Team: Hungarian\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=ISO-8859-2\n"
"Content-Transfer-Encoding: 8-bit\n"

msgid "Welcome to the program %s!"
msgstr "Köszönti Önt a(z) %s program!"
```

Az alábbiakban két `Makefile` olvasható. Ha van rendszergazdai jogosultságunk, használhatjuk az elsőt, máskülönben csak a második fog működni. A beljebb kezdett sorokat tabulátorral kell kezdeni.

```
all: ize ize.mo

ize: ize.c
      gcc -o ize ize.c

ize.mo: hu.po
      msgfmt -o
/usr/share/locale/hu_HU/LC_MESSAGES/ize.mo hu.po
```

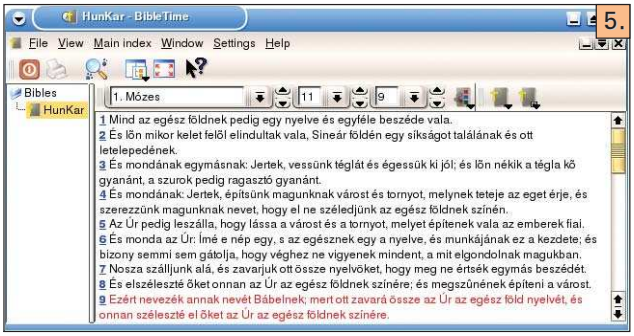
```
all: ize ize.mo

ize: ize.c
      gcc -o ize ize.c -D HELYBEN

ize.mo: hu.po
      msgfmt -o
      ➥$(HOME)/hu_HU/LC_MESSAGES/ize.mo hu.po
```

Ezek után adjuk ki:

```
$ make
$ ./ize
```

Ekkor a

```
Köszönti Önt a(z) ize program!
```



5.

szöveget kell látnunk. Ha a későbbiekben akár a *.c*, akár a *.po* fájlt módosítjuk, egyszerűen csak a `make` parancsot kell újra kiadnunk, amivel *ize* és *ize.mo* nevű bináris fájljaink önműködően frissülni fognak.

Többnyelvűség

A szép az egészben az, hogy a programunk innentől kezdve többnyelvű. Az egyes hozzáférhető nyelvek közül a program indítása előtt választhatunk:

```
$ LANG=de_DE; ./ize
```

Így például németül futna a programunk, ha a megfelelő könyvtárban megvolna az *ize.mo* fájl német nyelvű változata (mivel nincs, a feliratok angol nyelvűek lesznek).
Próbáljuk németül elindítani a Midnight Commandert, a Gimpet pedig spanyolul és csehül.
Mostantól más nemzetiségű fejlesztők, fordítók is csatlakozhatnak projektünkhöz, ha a *hu.po* fájlt például *de.po*-ra nevezik át, s az `msgstr` sorokat a megfelelő német fordításokkal helyettesítik. (Jobb, ha létezik egy központi *messages.pot* sablonfájl, amelyben az `msgstr` sorok üresek, s így a német fordítónak nem kell még a magyar fordítások kitörlésével is bajlódnia.)
Ha egy program magyar fordításával elégedetlenek vagyunk, az eredeti angol szövegeket úgy jeleníthetjük meg, ha a `LANG` változót `C`-re kapcsoljuk át (az `en_EN` érték ugyanúgy megfelelő).

Tippek és trükkök

A nemzetköziesítés éles használata számos gyakorlati kérdést vet fel. Ilyenek például a következők:

- Hogyan tehermentesíthető a fordító abban az esetben, ha a már lefordított szövegek időről időre részben (1–2 karakterben, szóban) megváltoznak? Azaz van-e mód arra, hogy ezekben az esetekben a fordítónak ne kelljen minden megváltoztatott szöveget teljes egészében újrafordítania?
- Megoldható-e, hogy a C nyelvű programban előforduló szövegekből önműködően jöjjön létre a megfelelő sablonfájl? Azaz megoldható-e, hogy a fejlesztőkre kevesebb karbantartói munka háruljon, vagyis a *messages.pot* fájlt robot hozza létre?
- Hogyan követhető nyomon, hogy a fejlesztők mely új szövegeket adták hozzá a program újabb változatához, és a fordító hogyan értesülhet a leggyorsabban arról, ha új szövegeket kell lefordítania?
- Megoldható-e, hogy már létező C-programokat gyorsan, önműködően nemzetköziesítsünk?

Mindezeket a kérdéseket (és számos továbbit) a gettext fejlesztői nagyrészt megoldották, és több segédprogramot bocsátanak rendelkezésünkre. Ilyen például az msgmerge program, amellyel a *hu.po* fájlt frissíthetjük az új változatú *messages.pot* fájl új üzeneteivel (esetlegesen arra is figyelve, hogy bizonyos szövegek időközben megváltoztak); vagy az xgettext, amellyel a C (vagy számos egyéb, például Java, Python vagy PHP) nyelvű programból az összes angol nyelvű szöveget „kirobbanthatjuk”, s azokból sablonfájlt szervezhetünk. Hasznos eszköz a gettextize is, amivel egy C nyelvű program könnyen átszabható úgy, hogy támogassa a gettext szabványt. A gyakorlatban a fejlesztők gyakran CVS-rendszert is bevonnak a programozói munkába, hogy a különböző változatok közötti eltéréseket könnyebben átláthassák (a nemzetköziesítéstől függetlenül is).
A fenti kérdésekre minden egyes program fejlesztésekor valamiféle választ szokás adni, de ez a projektektől függően más és más lehet. A megoldások többnyire a gettext használati utasításának ajánlásait követik (ám bizonyos részletekben gyakran el is térnek attól). Lássunk egy példát! A PostgreSQL adatbáziskezelő 7.3-as változatához egy éve új felügyeleti program készül pgAdmin3 néven. A nyelvi fordítás a fejlesztéssel párhuzamosan zajlik, azaz a fordítóknak rendszeresen új sablonfájlokat kell összefésülniük az általuk szerkesztett *.po* fájllal.
A nyelvi koordinátor a ➲ http://snake.pgadmin.org/pgadmin3/translation.php címen (*4. kép*) időről időre frissíti a sablonfájlt, amit a 26 fordító mindegyike, például a poEdit program *Catalog* menüjének *Update from POT file* lehetőségével fűzhet össze a saját *.po* fájljával. Ezután az új és a megváltozott szövegek honosításának megtörténtével az újonnan kapott *.po* fájlt vissza kell küldeni a koordinátorhoz, aki a program hivatalos forráskódjában (CVS-rendszeren) is rögzíti a változtatásokat. Ezt az eljárást mindannyiszor meg kell ismételni, ahányszor a sablonfájl módosul. (A gyakorlatban a fejlesztők mindig előbb járnak, mint a fordítók, így fordulhat elő, hogy a magyar fordítás szinte egyetlen programnál sem százszázalékos.)

A jövő

A gettext olyannyira jól bevált, hogy számos programozási nyelv alapértelmezésben támogatja a használatát. Ennek ellenére a bábeli zűrzavar teljes megoldása még hosszú távon is lehetetlen feladatnak tűnik. Évtizedek kemény munkájának tapasztalata alapján úgy látszik, hogy a fordítási munkafolyamat csak részben tehető önműködővé. Még egy számítógépes program fordítása során is sokszor szinte művészi tehetség kell, hogy a szövegeket anyanyelvünkön szépen, pontosan, csattanósan fogalmazzuk meg. Emiatt a fordító feladata fáradságos, hosszú, sziszifuszi munka lehet. Ezt mindenesetre kárpótolhatja az a tény, hogy a magyar fordításon felnőtt járatlan felhasználók kényelmesebben, nagyobb örömmel dolgoznak majd az adott programmal. Így a fordító közvetve igen sok új honfitárs felhasználót nyerhet meg nemcsak az adott programnak, hanem a Linuxnak is.

Tipp: töltsük le az internetről a BibleTime programot, és telepítsük mellé Károli Gáspár Biblia-fordítását (HunKar.zip). Keressük ki azt a részt Mózes 1. könyvéből, amelyik Bábel tornyának történetét meséli el (*5. kép*).

Kovács Zoltán
(kovzol@math.u-szeged.hu)
Tanársegéd a Szegedi Tudományegyetem Bolyai Intézetében az Analízis Tanszéken, matematikát és számítástechnikát tanít óraadóként a szegedi Radnóti Miklós Kísérleti Gimnáziumban.

Operációs rendszerek (14. rész)

Sorozatunk jelenlegi témája a lemezterület-kezelés, a fájlrendszerek megbízhatósága, egységessége és hatékonysága lesz.

Cikksorozatunk előző részében (Linuxvilág július, 62. oldal) megismerkedtünk a fájlkezelés alapfogalmaival. Most már nyugodtan belevághatunk a megvalósításba is. Mivel a fájlokat mágneslemezen tároljuk, a rendszer fejlesztőinek tulajdonképpen csak a lemez területének kezelésével kell megbirkózniuk, ami – mint hamarosan kiderül – elég nagy kihívás.

A lemezek kezelésével sorozatunk egyik korábbi részében már foglalkoztunk, amikor a blokkos beviteli–kiviteli eszközöket tárgyaltuk. Érdemes az ott leírtakat átismételni, mivel léptennyomon hivatkozni fogunk rájuk.

Minket most nem az eszközök érdekelnek, nem is fontos, hogy milyen lemezről van szó. A fájlkezelés nem más, mint a lemezterület-kezelés tudománya. Az a fontos, hogy milyen elv alapján tároljuk állományainkat a lemezen, úgy, hogy a tartalmukat bármikor visszaolvashassuk, felülírhassuk vagy letörölhessük. Ezenkívül az is nagyon fontos, hogy az egész folyamat gyors legyen.

A lemezterület kezelése sok tekintetben hasonlít a memóriáéhoz, csak még annál is bonyolultabb. Közös bennük, hogy itt is egy meghatározott méretű szabad területtel kell gazdálkodnunk olyan módon, hogy a lehető leggazdaságosabban ki tudjuk használni. Ugyanakkor ebben az esetben felmerül egy olyan tényező is, ami a memóriánál nem volt számottevő. Ez pedig nem más, mint az idő. Ha a lemezről be szeretnénk olvasni egy bájtot, akkor először is az olvasófejet úgy kell mozgatnunk, hogy az pontosan a felett a szektor felett legyen, amelyik a kívánt bájtot is tartalmazza. Egy szektor beolvasásának az ideje nem számottevő, de a fej mozgatása annál inkább. Míg a memória összes részét ugyanannyi idő alatt (gyakorlatilag azonnal) el tudtuk érni, addig a lemezek esetében a végrehajtási idő attól függ, hogy a fej éppen hol tartózkodik. Egy jó fájlrendszernél nem elég tehát, hogy gazdaságos tárolást nyújt, hanem az is fontos, hogy úgy helyezze el az adatokat, hogy például egy fájl eléréséhez a lehető legkevesebbet kelljen mozgatni a fejet.

Lássuk, milyen lehetőségek közül választhatunk! A virtuális memória esetében a szakaszolás és a lapozás volt a két meghatározó módszer, a lemezeknél is használhatunk valami ezekhez hasonlót. Például az állományokat tárolhatjuk folytonosan, azaz egybefüggő lemezterületen. Ez elsőre nagyon csábító lehet, mivel a fájlok olvasása rendkívül egyszerűen és gyorsan megoldható, és a fejet is csak egyszer kell megmozdítanunk.

Rögvest megváltozik a helyzet, amikor egy állomány növekedésnek indul. A memória esetében ilyenkor nem volt semmi gond, a szakaszt csak át kellett egy másik memóriaterületre költöztetni, ahol nagyobb szabad hely állt rendelkezésre. Csakhogy a memóriában történő mozgatás szinte semmi időbe se kerül, míg a lemezekről ez nem mondható el.

Célszerűbb, ha minden állományt egyenlő méretű blokkokra osztunk fel, amelyeknek természetesen nem kell szigorúan egymás mellett lenniük, akár össze-vissza is elhelyezkedhetnek a lemezen. Itt nem okozhat gondot a fájl méretének növekedése, viszont egy fájl teljes tartalmának a beolvasásakor valószínűleg több fejmozgatásra is szükség lesz. Mindenesetre a legtöbb fájlrendszer ezt a sémát használja.

A blokkméret megválasztása

Az odáig rendben van, hogy a fájlokat egyenlő részekre osztjuk, de vajon mekkorákra? Mivel a merevlemezek cilinderekre, sávokra, illetve szektorokra osztottak, megpróbálhatjuk ezeket választani az állományok helyfoglalási egységének.

A cilindert azonban semmiképp sem érdemes választani, mivel túl nagy. A nagy blokkmérettel az a gond, hogy sok kis állomány esetén rengeteg szabad kapacitás megy veszendőbe. Főleg, ha egy olyan rendszerről van szó, amelyik rengeteg apró fájllal dolgozik (például jellemzően ilyenek a Unixok).

Nagy a cilinder? Semmi gond, nézzük a szektort, ami sokkal kisebb. Valójában a szektor nemcsak kicsi, hanem túl kicsi ahhoz, hogy eszményi blokkméretnek választhassuk. Ez ugyan a lehető legkevésbé pazarló megoldás, viszont a nagy állományok rengeteg kis blokkból állnának, s ezek valószínűleg a lemezen egymástól jó távolra helyezkednének el. Egy ilyen állomány beolvasása a sok pozícionálás miatt próbára tenné a felhasználók türelmét.

Egyből két tanulságot is levonhatunk. Az első, hogy helyfoglalási egységet semmiképp sem a lemez „fizikai” egységei alapján érdemes választani. A másik pedig az, hogy minél kisebb a blokkméret, annál jobb a tárolási hatékonyság, viszont a fájlrendszer időhatékonysága annál rosszabb. Ez az állítás fordítva is igaz.

Mekkora lehet vajon a tökéletes blokkméret? Hát valahol a nagyon nagy és a nagyon kicsi között. Ez a nyilván megegyezésen alapuló választás nagymértékben függ a rendszertől, illetve attól, hogy kik használják. A legújabb kutatások mindenesetre azt mutatják, hogy a felhasználók átlagosan egyre nagyobb állományokat birtokolnak, tehát minden jel arra utal, hogy a nagyobb blokkméreteké a jövő.

Szabad blokkok

Amikor új állományt hozunk létre, vagy egy régihez újabb dolgokat írunk hozzá, akkor a rendszernek szabad blokkot kell foglalnia. Ehhez azonban fontos tudnia, hogy az összes blokk közül melyik szabad, és melyik nem.

Erre két elterjedt módszer létezik. Az első esetben a szabad blokkok címeit is blokkokban tároljuk, és ezeket egy láncolt listába fűzzük. Ez az úgynevezett láncolt lista, amit úgy képzelhetünk el, hogy van egy blokk, amelyben csak szabad blokkok címeit találjuk. Az utolsó helyen lévő cím azonban nem egy szabad blokk címe, hanem a következő olyan blokké, amelyik szabad címeket tartalmaz.

A másik az úgynevezett bittérképes módszer. Itt minden blokkhoz egy bitet rendelünk, ami azt jelzi, hogy szabad-e vagy sem.

Mindkét módszernek akadnak hátrányai. Vegyünk egy

1 GB-os merevlemezt 1 KB-os blokkmérettel – ez azt jelenti, hogy körülbelül egymillió blokkot tartalmaz. Láncolt listás megoldáskor, ha a teljes lemez üres, körülbelül négyezer blokkot (4 MB-ot) kell szánnunk az üres blokkok tárolására (32 bites blokkcímek esetében). A bittérképesnél az egész csak egymillió bitet, azaz pontosan 128 KB-ot foglal el.

A bittérképes megoldás jobbnak tűnhet, mint a láncolt listás. Ez azonban csak akkor igaz, ha lehetőség nyílik az egész bittérkép memóriában tárolására. Ma már memória ugyan sok van, de például egy 80 GB-os lemez esetében ez már 10 MB-unkba fog kerülni. Ez már elég nagy ahhoz, hogy ne tartsuk az egész bittérképet a memóriában.

Tegyük fel, hogy a memóriában mindig csak egy blokknyi bittérképet tartunk. Ha szabad blokkra van szükségünk, de a bittérképben nincs ilyen, akkor a rendszernek a bittérkép egy másik szeletét kell beolvasnia. Azt azonban semmi sem garantálja, hogy abban már találunk szabad blokkokat. Ha a merevlemezünk már csaknem tele van, azaz kevés a szabad blokk, megeshet, hogy szinte az egész bittérképet végig kell olvasnunk. A láncolt listás megoldásnál azonban csak egyetlen blokkal kell ezt megtennünk, és máris nyertünk újabb 255 szabad blokkcímet.

Ne felejtsük el azt sem, hogy a láncolt listás megoldásnál a lemez betelítésével csökken azon blokkok száma, amelyet a szabad blokkok nyilvántartására kell használnunk. Az 1 GB-os merevlemez esetében a bittérképnek mindig fent kell tartanunk szabad 128 kilobájtot, míg a láncolt listásnál egyet sem, feltéve, ha csurig töltjük merevlemezünket.

A fájlrendszer hatékonysága

A merevlemezek átlagosan százezerszer lassabbak, mint a memória. Ez az érték attól is függ, hogy a fej éppen hol tartózkodik, illetve mennyi adatot szeretne a felhasználó kiolvasni belőle. A memóriát például bájtonként, egyes rendszerekben szavanként címezhetjük, lemezek esetében azonban mindig az egész blokkot be kell olvasnunk, illetve ki kell írnunk. Ez nem szerencsés akkor, amikor csak egy-két bájtra lenne szükségünk.

Ezért a lemezterület kezelésnél előtérbe kerül a hatékonyság fogalma, azaz ki kell találnunk olyan módszereket, amivel a lemezműveleteket némileg felgyorsíthatnánk. Erre kétféle, ám együttesen használható megközelítés is létezik: egyrészt a lehető legkevesebbszer forduljunk a lemezhez, másrészt, ha már meg kell tennünk, akkor csökkentsük a lemezműveletek idejét. Az első célkitűzést egy gyorsítótár (cache) bevezetésével megvalósíthatjuk. Egy szektor beolvasását ugyan nem tudjuk felgyorsítani, ha viszont úgy rendezzük a blokkokat a lemezen, hogy a fejet ne kelljen minden kérés alkalmával pakolgatni, akkor a második feladatot is teljesíteni tudjuk.

A gyorsítótárnál alkalmazott elv sok tekintetben hasonlít a lapozott memóriáéhoz. Ha egy keresett blokk bent csücsül a gyorsítótárban, akkor a kérést anélkül ki tudjuk szolgálni, hogy a lemezhez fordulnánk. Ha nincs, akkor a gyorsítótárból egy blokkot kiválasztva ki kell írnunk a lemezre, majd betenni a helyére azt, ami éppen kell nekünk. Ehhez előbb ki kell választanunk, hogy melyik legyen az a blokk, amelyiknek távoznia kell. Ennek eldöntésére használhatjuk a lapozási algoritmusok valamelyikét, például az LRU-t vagy a FIFO-t.

A helyzet azonban nem ilyen egyszerű. Ha „színtiszta” lapozási algoritmusokat (például az LRU-t) használunk, akkor könnyen kerülhetünk kellemetlen helyzetbe. Képzeljük el azt a helyzetet, amikor módosítunk egy, a fájlrendszer szempontjából alapvető fontosságú blokkot, például olyat, ami fájlleírót (inode) tartalmaz. A módosítás a lemezen csak akkor könyvelődik el, ha az algoritmus a gyorsítótárból ezt a blokkot takarítja ki. Ha eközben esetleg egy rendszerösszeomlás következik be (vagy áramszünet, vagy hasonló katasztrófa), a fájlrendszerünk sérülni fog.

Ezt orvosolhatjuk akképpen, hogy bizonyos blokkokat (amelyek kiemelt szerepet játszanak a fájlrendszer életében) egyből kiírunk a lemezre, vagy az LRU-lánc elejére teszünk, tehát a következő cserénél az a blokk lesz az, amelyik kiíródik.

Az adatblokkokat csak akkor érdemes cserélnünk, ha várhatóan rövid időn belül nem lesz rájuk szükségünk. Az úgynevezett csonka adatblokkok mindig az LRU-lánc végére kerülnek, mivel ezek hamarosan kellenek, hiszen valószínűleg valamelyik alkalmazás írni fog bele.

Előfordulhat olyan eset, hogy egy adatblokk sosem kerül az LRU-lánc elejére, azaz nem kerül kiírásra. Jellemzően ilyen eset, ha valaki szövegszerkesztővel dolgozik. Például e cikk szerzője e sorok írásakor már harmadik órája szenved a cikkével, és mivel más lemezműveleteket végrehajtó programot nem futtat, a dokumentumot tartalmazó blokk minden bizonnyal nem kerül ki a gyorsítótárból. Ha a szerző egy után megtervezett operációs rendszert használna, és esetleg áramszünet következne be, akkor minden bizonnyal háromórányi munkája veszne el. Ezen még az sem segítene, ha az önműködő mentési szolgáltatás be lenne kapcsolva.

Szerencsére a szerző rendszerében (és minden Unixban) van egy `SYNC` nevű rendszerhívás, ami az összes gyorsítótárban lévő blokkot kiírja a lemezre. Ez a rendszerhívás mindig végrehajtódik, amikor egy lemezegység leválasztására készülünk, de ez még kevés. A `SYNC`-nek akkor van igazán haszna, ha valaki folyamatosan végrehajtja. Ezért az összes Unix induláskor egy folyamatot indít el (amelynek általában `update`, vagy rendszertől függően valami hasonló neve van), ami nem tesz mást, mint hogy félpercenként meghívja a `SYNC`-et. Rendszerösszeomláskor itt is bekövetkezhet némi adatvesztés, de csak annyié, amennyit az utolsó 30 másodpercben alkottunk.

Létezik azonban olyan rendszer is, amelyik minden módosított

blokkot egyből kiír a lemezre (ez az úgynevezett írásáteresztő gyorsítótár). Ilyen például az MS-DOS. Ezért van az, hogy DOS alatt mindenféle komolyabb következmény nélkül cserélgethetjük a hajlékonylemezeinket, míg például Linux alatt először le kell azt választanunk.

Ennek a módszernek az a hátránya, hogyha karakterenként írjuk az állományt, akkor annyi lemezműveletet kell végrehajtani, ahány bájtot kiírunk. Ez sokkal lassabb, mintha először összegyűjtenénk a változásokat a gyorsítótárba, majd egy SYNC rendszerhívás következtében egyetlen lemezművelettel az egészet kiírnánk. (Ezért, akik DOS alá fejlesztettek, gyakran programon belüli gyorsítótárat használtak. Ennek az volt a módja, hogy a kiírandó karaktereket először egy tömbbe tették, és ha az megtelt, akkor az egész tömböt kiíratták, majd indulhatott az egész elölről.)

Meg kell jegyeznünk azonban, hogy az MS-DOS-nak ez nem tervezési hibája, valójában kifejezetten ilyenre alkották. Ugyanis ezt a programot a személyi számítógépekre tervezték, még azokban az időkben, amikor a merevlemez nem volt alapvető kellék, és mindenki cserélhető lemezegységeket használt. A Unix azonban olyan gépeken fejlődött, amik nem éppen hajlékonylemezekről indultak, így ott volt értelme az írási műveletek gyorstárazásának is. Ma már nyilván nem a DOS példája a követendő.

A gyorsítótáron kívül fájlrendszerünk hatékonyságát úgy is növelhetjük, hogy valahogy csökkentjük a fej mozgását. Ezt úgy érhetjük el, hogy azokat a blokkokat, amelyekre nagy valószínűséggel egymás után fognak hivatkozni, a lehető legközelebb helyezzük el egymáshoz (még jobb, ha egy cilinderen is vannak).

Ezt úgy érhetjük el, hogy amikor szabad blokkokat kell foglalnunk, akkor amennyiben lehetséges, mindig egymás utáni blokkokat foglaljunk. Bittérképes nyilvántartás esetén, ha az egész bittérkép a memóriában helyezkedik el, ez egyszerű feladat. Láncolt listásnál sokkalta nehezebb.

A fájlleírókat használó fájlrendszerek esetében azonban egy másik gond is felmerül: még a legkisebb állomány eléréséhez is legalább két blokkot be kell olvasni. Amennyiben a fájlleírók a lemez elején lennének, a fejet általában a lemez közepére kellene pozícionálni. Javíthat a helyzeten, ha a fájlleírókat a lemez közepén helyezzük el, így átlagosan nézve a fej csak feleannyi távot fog befutni.

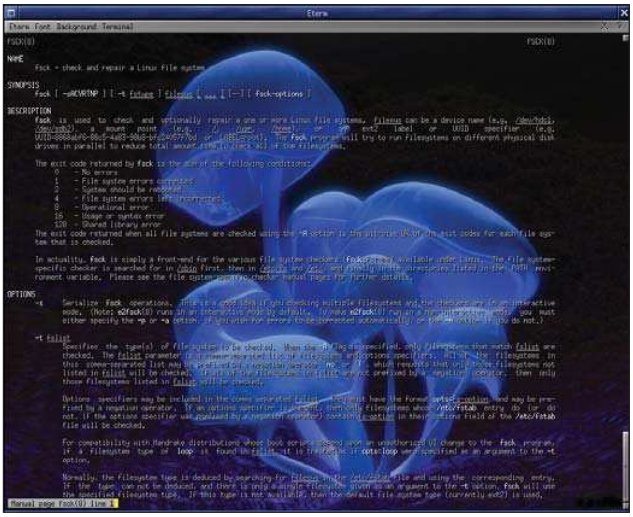
Fájlrendszer-ellenőrzés

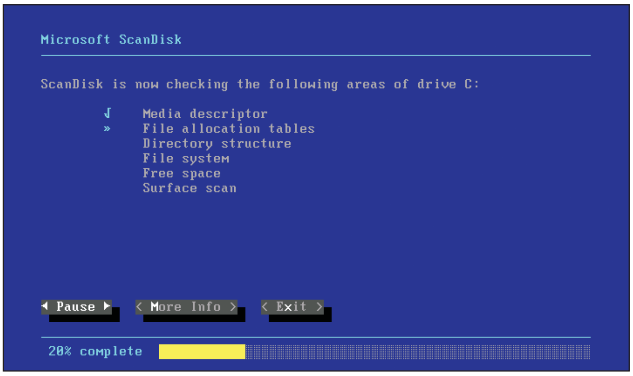
Minden igyekezet ellenére is előfordulhatnak olyan szerencsétlen esetek, amikor úgy hal meg a rendszerünk, hogy még maradt kiíratlan módosított blokk. Ha ez a blokk netán egy fájlleírót vagy egyéb fontos adatot tartalmazott a fájlrendszerről, akkor a fájlrendszer sérül, más néven inkonzisztens állapotba kerül.

Az inkonzisztens fájlrendszer nem biztos, hogy használhatatlan, lehetséges, hogy csak egy-egy blokk válik elérhetetlenné stb. Mindenesetre nem jó dolog, ha a fájlrendszer nem egészséges, ezért érdemes valamiféle inkonzisztencia-ellenőrző és -javító programot futtatni. A továbbiakban most egy ilyen alkalmazás működésének a rejtelmeibe pillantunk bele.

Az inkonzisztencia ellenőrzését végezhetjük blokkonként és fájlonként is. Először nézzük a blokkonkénti ellenőrzést!

Először is minden blokkhoz két számlálót rendelünk: az egyikben azt számoljuk, hogy az adott blokk hány fájlban fordul elő, a másikban pedig, hogy hányszor fordul elő a szabadlistában. Ezután végigmegyünk az összes fájlleírón, ennek alapján az összes blokkot, amelyhez fájl tartozik, elérhetjük. E blokkok számlálóját mindig megnöveljük eggyel. Ha ez kész, akkor jöhet a szabadlista (illetve a bittérkép) végigpásztázása. Ha egy blokk előfordul, akkor annak a másik számlálóját kell növelni. Ezzel a vizsgálat be is fejeződött: fájlrendszerünk akkor és csak akkor lehet összefüggő, ha az összes blokk számlálóinak értéke közül pontosan csak az egyik 1.





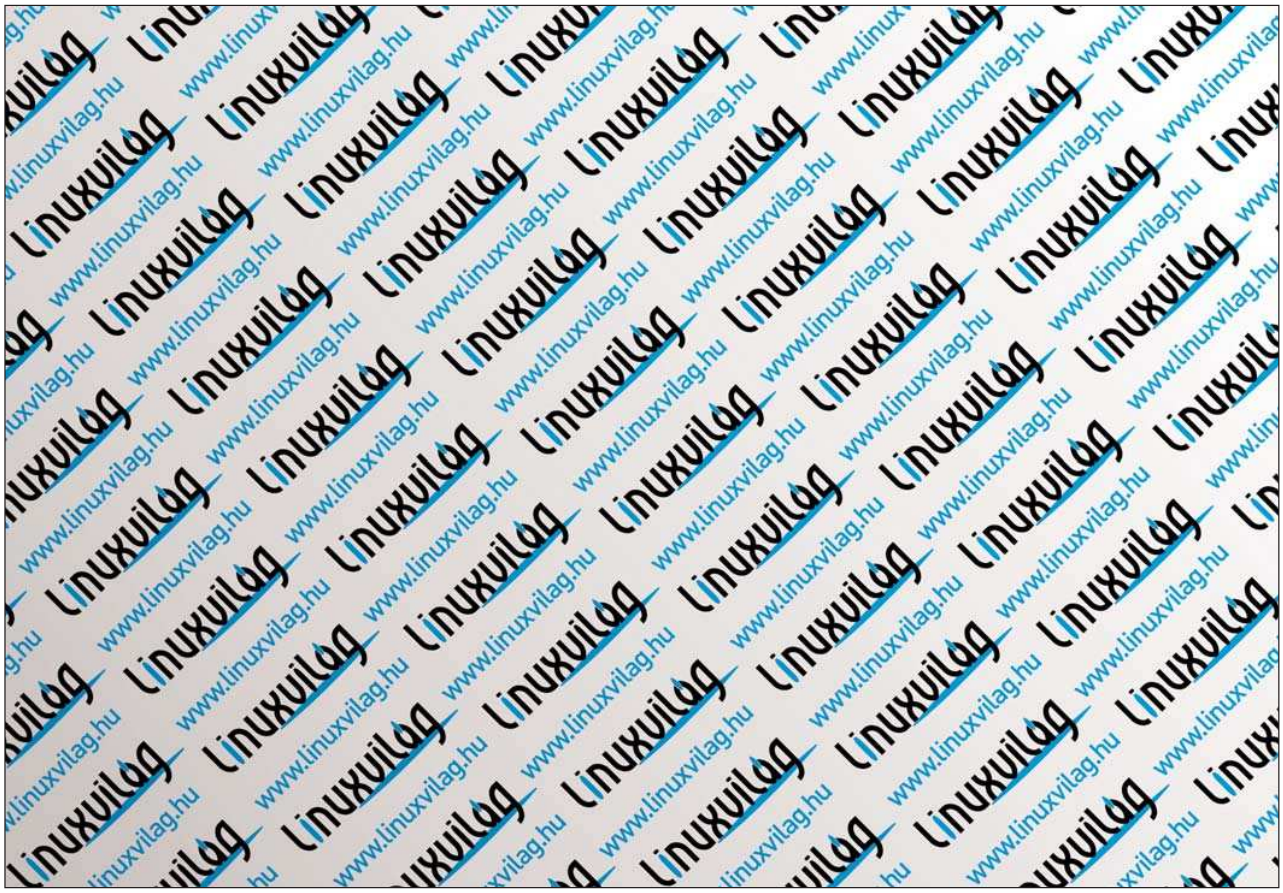
Ez logikus, mivel ha mindkét számláló nulla lenne, akkor az olyan, mintha az a blokk nem is létezne. Ebben az esetben hiányzó blokkról beszélünk. Ez nem jelent súlyos veszélyt a fájlrendszerre nézve, de kapacitásveszteséggel jár. A hiba szerencsére könnyedén javítható, csupán fel kell venni a szabadlistába. Az sem túlzottan egészséges, ha egy blokk kétszer szerepel a szabadlistában. Ebben az esetben a megoldás kulcsa az, hogy elölről építjük az egész szabadlistát, ügyelve arra, hogy az adott blokk már csak egyszer kerüljön bele.

Érdekesebb a helyzet akkor, ha azt kapjuk eredményül, hogy egy blokk több fájlhoz is tartozik. Ez nagyon veszélyes lehet a fájlrendszerre nézve, mivel ha letörlünk egy olyan állományt, amelyikben ez a blokk szerepel, akkor a blokk egyszerre lesz szabad is, meg foglalt is. Ha a másik ilyen állományt is letöröljük, akkor a blokk „kétszeresen" lesz szabad. Ezt a hibát ezért mindenképpen javítanunk kell.

A bevált módszer a következő: foglalunk egy új blokkot, ami a sérült blokk tartalmával töltünk fel. Ezután a sérült blokkot kiszedjük a második fájlból, a helyére pedig beteszszük az újonnan létrehozott blokkot. Valamelyik állomány tartalma minden bizonnyal sérülni fog, de nézzük a jó

oldalát: fájlrendszerünk következetessége helyreállt.
A fájlok szerinti összefüggés ellenőrzésekor blokkok helyett az állományokon megyünk végig. Pontosabban belenézünk az összes könyvtárba, és megszámoljuk, hogy egy adott fájlleíróhoz hány állomány tartozik. (Minden fájlleíró tartalmaz egy kapcsolatszámláló nevű változót, ami megmondja, pontosan hány állomány is tartozik hozzá.) Ezután belenéz a fájlleíróba is. Ha ott több, netán kevesebb állomány szerepelne, mint amennyi a könyvtárakon végzett ellenőrzés eredménye, akkor baj van.
Ha a fájlleíró kapcsolatszámlálójának értéke nagyobb, mint amennyit mi számoltunk, az még a jobbik eset. Ha nem javítjuk, annyi történhet, hogy ha az összes olyan állományt kitöröljük, amelyik az adott fájlleíróhoz tartozik, akkor a fájlleíró maga nem szabadul fel, mivel a számláló értéke biztosan nagyobb lesz nullánál. Tehát helypazarlás áll fenn. Mindenesetre a kapcsolatszámláló csökkentésével a gond megoldható.
A legizgalmasabb azonban az az eset, amikor több állományt számoltunk össze, mint amennyiről a fájlleíró „tud". Ilyenkor ugyanis adatvesztéssel is számolnunk kell. Miért is? Tegyük fel, hogy letöröljük az egyik állományt. Ha a fájlleíró számlálója egy volt, akkor most nulla lesz, és felszabadul a fájlleírót tartalmazó blokk. A másik állományt és annak tartalmát soha többé nem fogjuk tudni elérni. Azért a helyzet korántsem ilyen súlyos, egyszerűen csak állítsuk a kapcsolatszámláló értékét annyira, amennyit mi számoltunk.
A mostani ellenőrzőprogramok e két módszert (a blokk és az állomány szerinti ellenőrzést) együtt használják, hatékonysági okból kifolyólag a két folyamat egymástól el sem választható. Ennek köszönhetően a fájlleírót csak egyszer kell átnyálazni.
Ezek a hibák, amelyek veszélyeztethetik a fájlrendszer összefüggőségét, általában mindig a felhasználótól független események miatt következnek be, például rendszerösszeomlás, áramszünet miatt. Igaz, akad egy kivétel: a rendszer helytelen leállítása. De most már tudjuk, mivel járhat egy ilyen cselekedet, így remélhetőleg senki sem fog ilyesmire vetemedni.
Nem számoltunk még azonban a felhasználó által okozott hibákról sem, ilyen például állományok véletlen letörlése, vagy a törlés parancs helytelen használata. Az ilyenek miatt nem lesz a fájlrendszer inkonzisztens, csak a felhasználó adatai tűnhetnek el, de ez elég ok arra, hogy létezzen valamilyen védelmi módszer. Nem minden fájlrendszer tartalmaz ilyet, és ha tartalmaz is, mind-mind különböző módon. Az MS-DOS például nem végezte el a tényleges törlést (azaz a blokkok felszabadítását) egészen addig, amíg volt más szabad hely a lemezen. Addig a felhasználónak lehetősége volt a megfelelő parancs segítségével visszahozni a letörölt állományokat.
A következő részben a biztonsággal foglalkozunk, pontosabban azzal, hogy az operációs rendszerek általában miként védik az adatainkat. Természetesen arról is szót ejtünk, hogy mindezt hogyan valósítják meg.

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



Héjprogramozás Linux alatt – az AWK nyelv (5. rész)

Héjprogramozásról szóló sorozatunk mostani részében az AWK sedégprogrammal és szövegfeldolgozó nyelvvel ismerkedünk meg.

Valószínűleg az olvasónak is feltűnt, hogy az előző részben bemutatott megoldás meglehetősen körülményes volt. A gondot alapvetően az okozta, hogy maga a héj nem rendelkezik olyan szolgáltatással, amivel egyszerűen hozzáférhetnénk egy táblázat elemeihez. Annak idején talán ez volt az egyik olyan felismerés, ami elvezetett az `awk` segédprogram és szövegfeldolgozó nyelv megalkotásához. (A furcsa név a szerzők neveinek kezdőbetűből állt össze: ***Aho, Weinberg, Kernighan***.)
Az `awk` – bár mérete általában nem haladja meg a néhány száz kilobájtot – a korszerű táblázatkezelők gyakorlatilag valamennyi szolgáltatásával rendelkezik. Az egyetlen – ám sokak számára igen lényeges – eltérés az, hogy ennek a programnak nincs grafikus felülete, vezérlésére ehelyett egy saját programozási nyelv szolgál.
Az `awk` nyelve számos ponton erősen hasonlít a C-re. Ennek megfelelően összes függvényét, vezérlési szerkezetét és egyéb szolgáltatását egyetlen cikkben lehetetlen bemutatni. Itt és most csak azokról az alapelvekről és lehetőségekről esik majd szó, amelyek a héjprogramokban a leggyakrabban felbukkannak.

Az awk működésének logikája

Az `awk` bemenete fájlból és a szabványos bemenetről egyaránt érkezhet. A feldolgozást vezérlő programot szintén meg lehet adni külön fájlban (ilyenkor a `-f` kapcsolót kell használnunk), de gyakoribb az a megoldás, amikor ezt is rögtön az `awk` kulcsszó után, egyszeres idézőjelek (aposztróf) közé zárva adjuk meg. Egy az `awk`-val kapcsolatos kódrészlet tehát sematikusan a következőképpen nézhet ki:

```
...bemenet... | awk 'BEGIN \
{ utasítások }
{ főprogram }
END { utasítások }'
```

Az első sor végén látható \ (visszaperjel) sorösszefűzést végez, vagyis azt jelzi a héj számára, hogy csupán az olvashatóság kedvéért írtuk a kódot több sorba. (Ügyeljünk rá, hogy a \ után már semmi nem állhat, még szóköz sem!)
A „főprogramban” található utasításokat az `awk` a bemenet valamennyi során önműködően végrehajtja, vagyis a program eleve tartalmaz egy rejtett (implicit) ciklust. (Emlékezzünk rá, hogy az előző részben a táblázat sorainak végigpásztázásához nekünk magunknak kellett ciklust írnunk.)
A `BEGIN` blokkban szereplő utasítások a sorok feldolgozásának megkezdése előtt hajtódnak végre, az `END` blokk tartalma pedig a feldolgozás után fut le.
És most következzék a lényeg: az `awk` feldolgozás előtt minden sort mezőkre bont. Alapértelmezésként mező az, amit mindkét oldalról legalább egy szóköz vagy tabulátor határol. A mezőkre a programban a `$1`, `$2` stb. szimbólumok segítségével egyenként lehet hivatkozni. A számozás egytől indul, de létezik a `$0` is, ami a teljes sort jelenti. (Figyelem, ezeknek a szimbólumoknak semmi közük a héjprogram parancssori kapcsolóihoz!)
Az `awk` programokban megadhatunk változókat, amelyekre itt a héj „szokásaitól” eltérően minden helyzetben egyszerűen a nevük leírásával lehet hivatkozni. Az `awk` az eddig használt `expr` paranccsal szemben lebegőpontos számokkal is képes műveleteket végezni, és minden olyan különleges függvényt is ismer, amit egy tudományos zsebszámológépen megtalálhatunk. Az eredmények kiíratására a `print` parancsot használhatjuk.

A sorok összegzése

Ennyi bevezető után lássuk, miként valósítható meg az `awk` segítségével egy táblázat sorait összegző függvény:

```
1: sorosszeg()
2: {
3:   cat $1 | awk \
4:   '{ osszeg=0
5:      for(i=1;i<=NF;i++) osszeg+=$i
6:      print osszeg}'
7: }
```

A `cat` parancs utáni `$1` szimbólum most is a függvény (és nem a program) első parancssori értékét jelenti, a 3. sor végén pedig az előbb említett, az olvashatóságot megkönnyítő sorösszefűző karakter látható. A pillanatnyi sor mezőin végig haladó `for` ciklus akár a C-program része is lehetne. A C nyelvet nem ismerők kedvéért talán érdemes megemlíteni, hogy az `i++` szimbólum az `i=i+1` művelet „rövidített” megfelelője, a `+=` műveleti jel (operator) pedig az `osszeg=osszeg+$i` művelet leírását teszi egyszerűbbé. Mindkét megoldás a C nyelvből ered.
A ciklus leállási feltételében szereplő `NF` az `awk` egyik önműködően frissülő belső változója, ami a feldolgozás alatt álló sor mezőinek számát tartalmazza. Gyakran szükséges ezenkívül még az `NR` belső változó, ez a pillanatnyi sor sorszáma (a számozás egytől indul).

Az oszlopok összeadása

A sorok összegzésénél nem volt szükség az `awk` programon kívüli ciklusra, mivel a végigpásztázásuk önműködő. Az oszlopok esetében ilyen automatizmus már nincs, így itt két ciklusra lesz szükségünk. A megoldás azonban még így is rövidebb és áttekinthetőbb lesz, mint az előző részben bemutatott.

```
1: oszloposszeg()
2: {
3:   oszlopszam=`head -1 $1 | wc -w`
4:   i=1
5:   while [ $i -le $oszlopszam ]
6:   do
```

```
 7:      cat $1 | awk -v oszlop=$i \
 8:      'BEGIN { osszeg=0 }
 9:             { osszeg+=$oszlop }
10:       END { print osszeg }'
11:      i=`expr $i + 1`
12:    done
13: }
```

A 3. sorban kiderítjük, hogy hány oszlopból áll a táblázat. Ehhez a `head` segítségével kiemelt első sorban megszámoljuk a szavakat. A következő érdekes momentum a 7. sorban látható. Itt az `awk -v` kapcsolója segítségével kívülről állítjuk be az oszlop nevű `awk` változó értékét. Ez fogja megmutatni, hogy éppen hányadik oszlop összegzésénél tartunk. A szabványos ki- és bemeneten kívül a `-v` kapcsoló szolgáltatja az egyetlen kapcsolattartási lehetőséget a héjprogram és az `awk` program között. Ezzel a módszerrel természetesen több változó tartalmát is beállíthatjuk, a `-v` kapcsolót azonban minden értékadásnál meg kell adni.
Magának az `osszeg` változónak a nullázása most a `BEGIN` blokkba került, hiszen ezt a műveletet csak egyszer akarjuk elvégezni. Hasonló okok miatt került az eredmény kiíratása az `END` blokkba. Maga a főprogram csupán egy összegzést tartalmaz.
A kész héjprogram most is a fenti részek egyszerű összemásolásával állítható elő, a parancssori kapcsolók kezeléséhez pedig ugyanazt az egyszerű `case` szerkezetet használhatjuk, mint a sorozat előző részében.

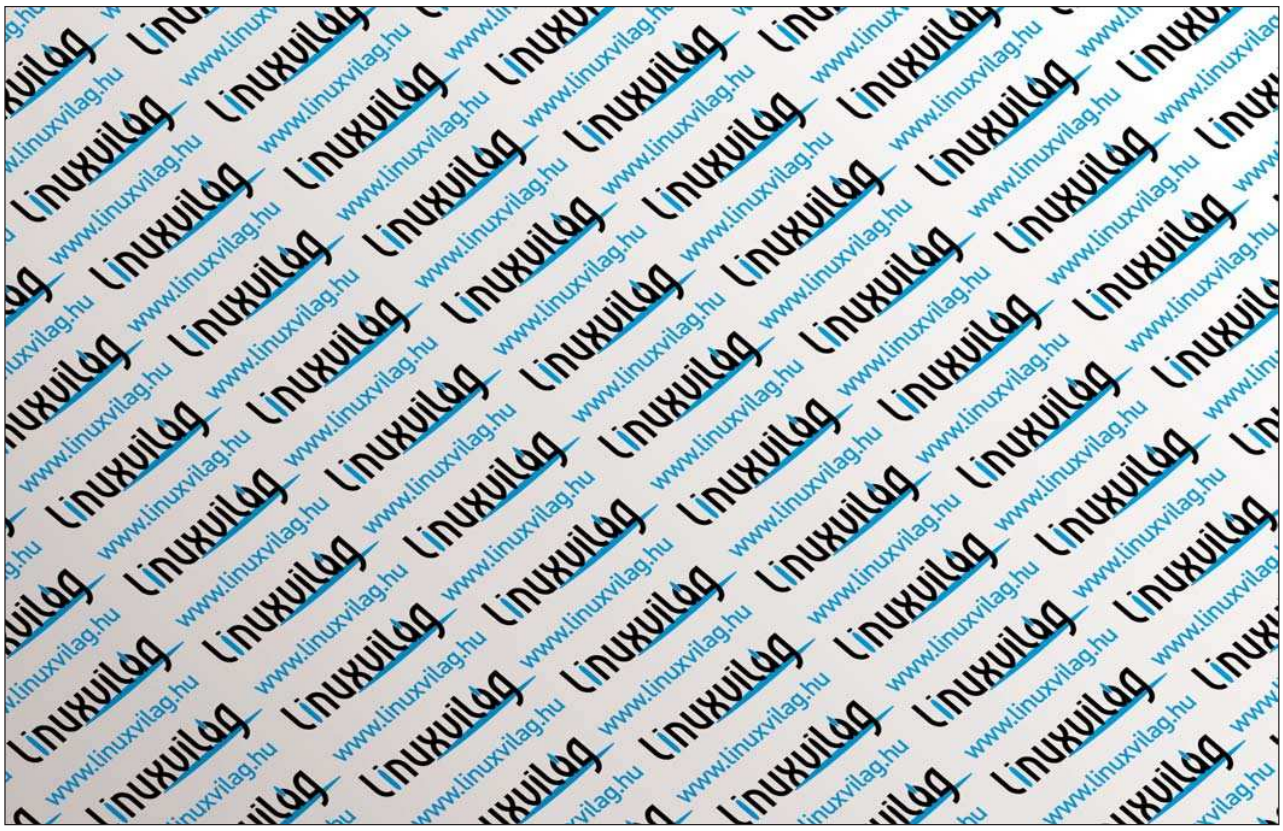
Szépészet awk-val

Befejezésül bemutatom az `awk` használatának egy másik lehetőségét is. Biztosan az olvasónak is feltűnt már, hogy milyen „rendszerető módon” vannak sorszámozva a sorozatban megjelent kódszövegek. Egy igazi Linux-felhasználónak természetesen eszébe sem jut, hogy ilyesmit kézzel tegyen meg – helyette megírja a következő nyúlfarknyi programot:

```
1: #!/bin/sh
2: cat $1 | awk \
3: '{ if (NR<10)
4:        { print " "NR": " $0 }
5:    else
6:        {print NR": " $0}
7: }'
```

Mint korábban említettem, az `NR` belső változó, amelynek a tartalma az éppen feldolgozott sor sorszáma, vagyis éppen az, amire nekünk szükségünk van.
Ha azt akarjuk, hogy az egy- és kétjegyű sorszámok utáni kettőspontok szépen egymás alatt sorakozzanak, az egyjegyűek elé egy-egy szóközt is be kell illesztenünk. Éppen ezt teszi a fenti döntési szerkezet igaz ága. (Figyeljük meg, hogy `awk`-ban a karakterláncok összefűzéséhez elegendő egyszerűen megszakítás nélkül egymás mellé írni őket.)
Van ebben a megoldásban egy kis tökéletlenség: akkor is beljebb tolja az egyjegyű sorszámokat, ha nincsenek kétjegyűek. Ennek megoldását az olvasóra bízom.

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



Az XML és a Perl (1. rész)

Az egyik legáltalánosabb adattárolási módszer alkalmazása a legnépszerűbb programozási nyelvek egyikével.

Ha programozási nyelvről beszélünk, egy olyan, a számítógép által értelmezhető szövegre gondolunk, ami egy alkalmazást ír le. Erre jó példa a Perl. Ugyanakkor meglepően sűrűn előfordulhat az is, hogy az adatunk válik olyan bonyolulttá, hogy külön nyelvre van szükségünk a leírásához. Igen egyszerű adatnyelvek is léteznek már. Először ezekről esik szó.

Adatnyelvek

Feltételezem, hogy némi gyakorlatra tettél szert a Perlben, és adatforrásként használtál már vesszővel elválasztott mezőket tartalmazó sorokat (más néven rekordokat) magába foglaló állományt. Léteznek ennél bonyolultabb nyelvek is, mint az, amelyikkel a `Makefile`-okban találkoztál. Ez egy program lefordításához szükséges adatokat tartalmazza. A szövegalapú adattárolás csúcsát bizonyos értelemben a Sendmail beállító-állománya jelenti. Ha eddig nem találkoztál vele, örülj neki, hogy megúsztad.

Az SGML

Hozzávetőlegesen harminc évvel ezelőtt vetődött fel egy teljesen általános adatnyelv létrehozásának az ötlete. Hamar be kellett látni azonban, hogy lehetetlen lenne az adattárolás területén minden elképzelhető célra egyetlen nyelvet felkészíteni. Így az utóbbi feladattal küzdő csoport új célt tűzött ki magának. Egy olyan metanyelvet kívántak létrehozni, amellyel már könnyedén készíteni lehet egy jelölőnyelvet egy valóságos dokumentumhoz. Ennek a törekvésnek az eredménye az SGML (Standard Generalized Markup Language), amely 1986-ban vált nemzetközi szabvánnyá. Az SGML neve egy picit megtévesztő, hiszen ez önmagában nem egy jelölőnyelv (markup language), hanem egy metanyelv jelölőnyelvek meghatározásához. Minden ilyen jelölőnyelvet SGML-alkalmazásnak hívunk, ami ismét megtévesztő lehet azoknak, akik az alkalmazás szót csupán egy program szinonimájaként ismerik. A legtöbb SGML-alkalmazás nagyon sokáig ismeretlen volt az amerikai katonaságon és az egyes közhivatalok berkein kívül. A 90-es évek elején azonban színre lépett egy jelölőnyelv, amire a média is hamar rákapott: ez volt a HTML (HyperText Markup Language). Minden SGML-alkalmazás hasonlít a HTML-re annyiban, hogy egyetlen elemből áll, ami elemeket tartalmaz, s ezek további elemekből, illetve egyszerű szövegből állnak. Az elemek elejét és a végét kezdő- és zárócímkék jelentik, amelyek relációjelek között szerepelnek. Nagy vonalakban így fest egy SGML-dokumentum. Az SGML nagyon tág fogalom, és írásmódja számos dolgot engedélyez, ezért olyan nehéz SGML értelmezőt írni. Egy SGML-értelmező ráadásul addig nem is tud megfelelően értelmezni egy dokumentumot, amíg nincs meghatározva a jelölőnyelv nyelvtana, a DTD (Document Type Definition) – de erről majd később ejtünk szót.

Az XML

Az SGML azt mutatta, hogy egy hozzá hasonló nyelv tökéletes választás lehetne a különböző programok közötti adatcsere megoldására. Hamar be kellett látni, hogy a hasonlónak egyben egyszerűbbnek is kell lennie. 1996-ban a World Wide Web Consortium (W3C) megbízott egy csoportot az SGML kistestvérének a kidolgozásával, ami elég egyszerű ahhoz, hogy adatcseréhez lehessen használni, különösképpen webes alkalmazások esetében. Ez volt az XML (eXtensible Markup Language). 1998 februárjában jelent meg az XML-ajánlás, amit a ➲ http://www.w3.org/TR/REC-xml címen olvashatsz el.

Az XML működése

Minden XML-dokumentum egy elhagyható bevezetőből (prolog) és egy ezt követő elemből, a gyökérelemből (root element) áll. Minden elem elejét egy kezdőcímke jelöli, ami egy kisebb jelből (<), egy névből, egy elhagyható tulajdonság–érték párokból álló listából, és egy záró nagyobb jelből (>) áll. Minden elem végét zárócímke jelöli, ami hasonlít a kezdőcímkére, de a kisebb jel után közvetlenül egy perjel (/) található, és nincs tulajdonság–érték listája. Az üres elemek azok, amelyek nem tartalmaznak további elemeket, illetve szöveget. Ezeket olyan módon írhatjuk le rövidebben, hogy elhagyjuk a zárócímkét, ugyanakkor a kezdőcímkében a nagyobb jel elé egy perjelet szúrunk be. Egy nagyon egyszerű XML-dokumentum így fest:

```
<dokumentum>Szia, vilag!</dokumentum>
```

Szándékosan használtam a magyar dokumentum szót, hiszen a címke nevét én határozom meg. Ebben a példában mindössze egyetlen gyökérelemünk létezik, a dokumentum, és csak egyszerű szöveget tartalmaz. Nincsenek tulajdonságai (attributum) – adjunk neki egyet!

```
<dokumentum tipus="pelda">Szia, vilag!
</dokumentum>
```

Most már `dokumentum` elemünknek van egy tulajdonsága, a `tipus`, ennek értéke a `"pelda"`. Az XML-ben minden értéket idézőjelek közé kell tenni, és minden tulajdonságnak kötelező értéket kell adni. Az érték nélküli tulajdonságok elfogadottak az olyan SGML-alkalmazásokban, mint a HTML, de az XML-ben nem.
Az XML-t létrehozó csapat ezt annak érdekében határozta meg, hogy az XML-értelmezők egyszerűbbek és ezáltal gyorsabbak legyenek.
Az eddigi példákban dokumentumaink csak egyetlen elemből álltak, és az is csupán szöveget tartalmazott (hivatalosabban karaktereket). Az XML igazi ereje azonban strukturáltságában rejlik: szövegen kívül az elemek további elemeket is tartalmazhatnak. Lássunk erre is egy példát!

```
<kozert>
        <csoport nev="zoldseg">
                <aru>krumpli</aru>
        </csoport>
```

```
        <csoport nev="gyumolcs">
                <aru>alma</aru>
                <aru>korte</aru>
        </csoport>
</kozert>
```

Itt a gyökérelemünk a `kozert` nevet kapta. Ez két `csoport` nevű elemből áll, ezek `nev` tulajdonsággal rendelkeznek. Az egyik a `zoldseg`, míg a másik a `gyumolcs` értéket kapta. Ezeken belül `aru` elemeket találunk, a fenti két típusnak megfelelően elrendezve. Az XML szépségét mutatja, hogy ezt a szerkezetet többféleképpen is feldolgozhatjuk, ahogyan a sorozat következő részeiben bőven láthatunk rá példát. A DOM felépítés például egy faelrendezésben tálalja számunkra az adatokat. Mivel a cikksorozat az XML és a Perl kapcsolatával foglalkozik, ez a bevezető nem szolgálhat minden részletre kiterjedő leírással az XML-t illetően. Ha további tájékoztatásra lenne szükséged, látogass el a ➲ http://www.xml.com oldalra.

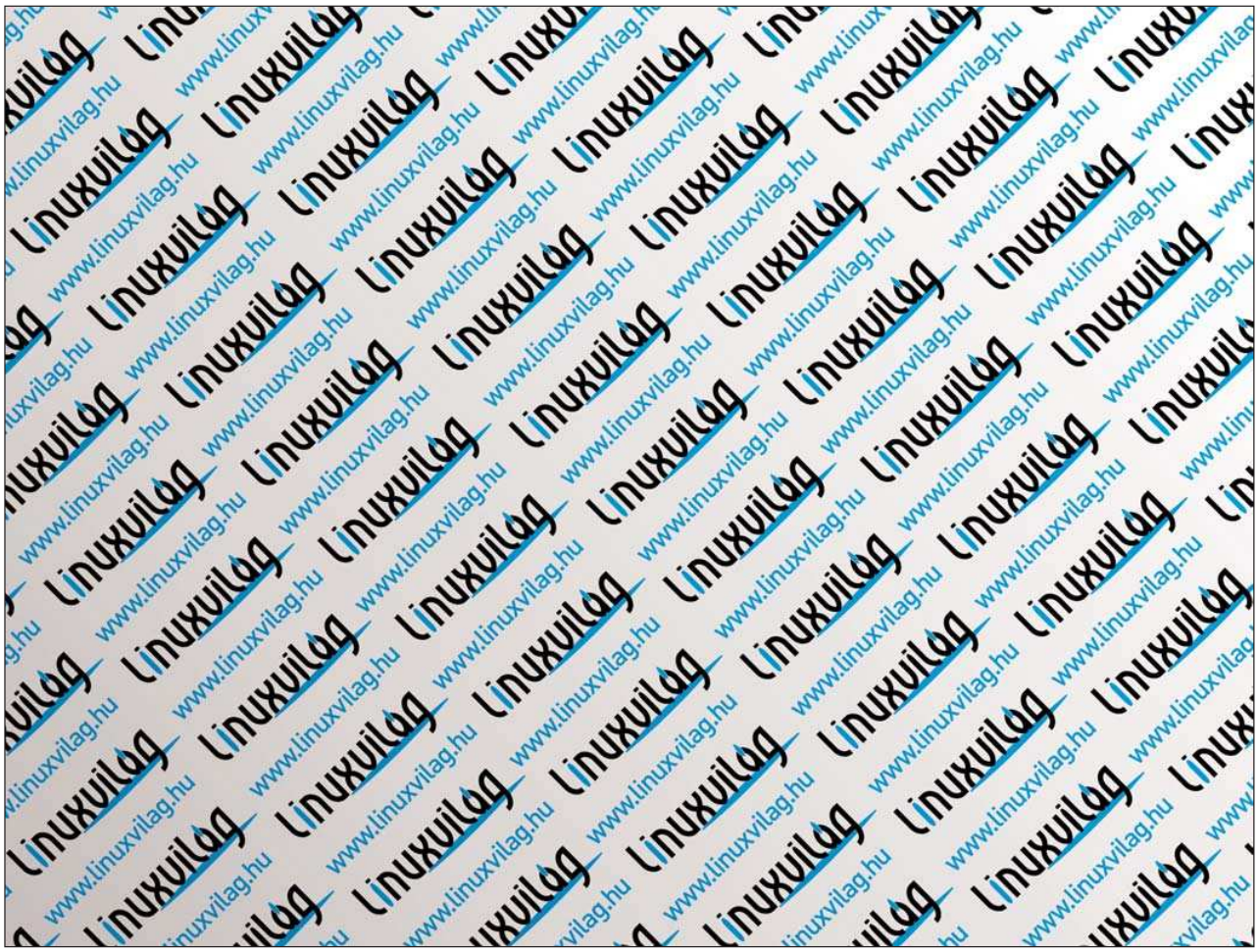
Mi nem az XML?

Számos félreértésből származó tévhit kering az XML-ről, és arról, hogy miért is hozták létre. Az XML azonban

- ***nem*** a HTML helyettesítője. Az XML egy metanyelv, míg a HTML tényleges nyelv, amit az SGML metanyelvvel határoztak meg. A HTML-nek előre rögzített elemei vannak, az XML-alkalmazások elemeit az alkotóik határozzák meg. Ugyanakkor a HTML-t az XML szabályai alapján is írhatjuk, és elképzelhető, hogy a HTML következő változata egy XML-alkalmazásként fog megjelenni.
- ***nem*** olyan nyelv, ami otthoni kiadványszerkesztést tesz lehetővé a weben.
- ***nem*** a relációs adatbázisok helyettesítője. Kis adatmennyiség esetén használhatunk XML-alkalmazást relációs adatbázist használó programok közötti adatcserére, de nem túl megfontolt döntés egy egész adatbázist így tárolni.
- ***nem*** egy mindenre használható svájcibicska tetszőleges nyelvek létrehozásához.

Létre lehetne hozni egy egész programozási nyelvet XML-ben, de ronda lenne és teljes mértékben rugalmatlan. Az XML olyan adatok ábrázolására alkalmazható jól, amelyek természetüknél fogva hierarchikusak, illetve bizonyos matematikai modellek esetében (mint a gráfok).
Szerintem az XML érdekes megközelítése az adattárolásnak, és megér egy próbát. Mindent ki kell próbálni, és ha már használtad, eldöntheted, hogy tetszett-e vagy sem.

Fülöp Balázs (xut@freemail.hu)
18 éves, imádja a Túró Rudit, a Debian Linuxot és a teheneket. Kedvenc írója Slawomir Mro□ek. Leginkább a számítógépes hálózatok biztonsága érdekli. A BME VIK műszaki informatikus szak hallgatója.



CMF-típusok

Egy tartalomkezelő csak testreszabás után használható igazán. Kezdjük munkánkat egy olyannal, ami elég hatékony ahhoz, hogy éppen úgy működjön, mint a szervezetünk.

Az előző hónapokban általánosságban foglalkoztunk a tartalomkezelő rendszerekkel (CMS), majd külön kiemeltük a Zope tartalomkezelő keretrendszerét (CMF). A Zope CMF olyan eszközt ad a fejlesztők kezébe, amellyel megalkothatják a saját tartalomkezelő rendszerüket. Természetesen az, aki már dolgozott CMS-rendszerrel, tudja, hogy felhasználás előtt még a kereskedelmi változatok legtöbbjét is alaposan át kell szabni és újra kell dolgozni. A Zope nemcsak egyszerűen az alapprogram árát csökkenti, de gazdag környezetet is kínál, amelyben könnyedén fejleszthetjük és ízlésünkhöz igazíthatjuk a CMS-t.

A CMF-oldalak létrehozása során (az oldal gazdájaként) dokumentumokat adhatunk a rendszerhez, illetve módosíthatunk és törölhetünk. Kattintsunk a könyvtártartalom hivatkozásra, majd bökjünk a *New...* gombra, és válasszuk ki, hogy milyen típusú dokumentumot szeretnénk, illetve milyen azonosítóval rendelkezzen, majd kattintsunk az *Add* gombra. Gépeljük be a segédadatokat (ez a cím, a leírás, a téma és a tartalomtípus), kattintsunk a *Change&Edit* gombra, gépeljünk be valami tartalmat, és már kész is vagyunk.

Igaz ugyan, hogy a már meglévő tartalomtípusok egy-egy egyszerű oldal esetében elegendőnek bizonyulhatnak, a kifinomultabb oldalakhoz általában egyedi változatokra van szükség: a saját típusokra. A CMF pontosan ennek a lehetőségét kínálja fel. Ebben a hónapban a CMF-be illeszthető típusokkal foglalkozunk, megtudhatjuk, hogy miképpen dolgozzunk velük, viselkedését hogyan szabjuk az igényeinkhez, milyen módon telepítsünk újabb elemeket, vagy akár hogyan készítsünk saját tartalmat kezelő egyedi típusokat.

A CMF-típusok

Az új típusok létrehozásának legegyszerűbb módja, ha a CMF beépített, webalapú típuskiterjesztés rendszerét használjuk. Ennek segítségével olyan új típusokat hozhatunk létre, amelyek tagfüggvényeiket, tulajdonságaikat, műveleteiket, megjelenítő sablonjaikat és ikonjaikat megoszthatják a többi típussal. Amikor a webalapú kiterjesztésrendszerrel új típust hozunk létre, az elemek bármelyikét megváltoztathatjuk, kivéve a tagfüggvényeket és a tulajdonságokat (properties). Más szavakkal, újonnan létrehozott típusunk akár egészen más kinézettel is rendelkezhet, mint a szülőtípusa, a viselkedése azonban továbbra is nagyon hasonló lesz.

A példa kedvéért vegyük a *types* eszközként ismert elemet, amihez a CMF-oldalunk kezelőfelületének *portal_types* elemére kattintva juthatunk el. Ha még nem adtunk meg CMF-oldalt a Zope alatt, könnyedén készíthetünk egyet a webalapú Zope-kezelőfelület jobb felső sarkában található *Add...* menü *CMF Site* pontjának kiválasztásával. Miután elkészítettük az oldalt, a kezelőfelületből a hozzátartozó ikonra kattintva jó néhány, csavarkulcsra emlékeztető ikonnal ellátott különböző beállításeszközt fogunk találni.

Amikor első ízben lépünk be a *types* (típusok) eszközbe, megfigyelhetjük a jelenleg megadott CMF-típusokat, többek közt a *folders* (mappák), *documents* (dokumentumok), *news items* (hírelemek), *links* (hivatkozások) és *topics* (témák) típusait. A kiválasztott típus nevére kattintva megnézhetjük, illetve megváltoztathatjuk a típusokhoz tartozó tulajdonságokat és műveleteket. Ha például a *File* tartalomtípus működését meg szeretnénk változtatni, kattintsunk a *File*-ra. Az oldal tetején új fülkészlet jelenik meg, amelyek közül csak a *properties* (alapértelmezett tulajdonságok) és az *actions* (műveletek) lesz az, ami nem része a hagyományos Zope-készletnek. Igazság szerint a tulajdonságok fül a szabványos Zope-fülek közt is megtalálható, de a CMF-típusok számos szokatlan tulajdonságnévvel is bírnak.

A megszokott szabványtulajdonságokon túl minden típus néhány, a működését befolyásoló egyéb tulajdonsággal is rendelkezik:

- *Icon*: karaktersorozat, ami azt írja le, hogy az adott típushoz milyen ikon jelenjen meg.
- *Product metatype*: a Zope-termék metanevét adhatjuk meg itt. A metaneveket az *Add...* menüpontban a Zope-kezelőfelületén (Management Interface) használjuk. A CMF hasonló *Add...* menüjében úgyszintén ezt a nevet találjuk.
- *Product name*: azt a Zope-terméket jelöli meg, amelyikben a CMF-típust meghatároztuk. Minthogy a *File* és a *News* elemtípusok már az eredeti CMF-telepítésben is benne vannak, itt a *CMFDefault* termék neve szerepel. Ami igaz is, hiszen ha belenézünk a */lib/python/products/CMFDefault* könyvtárba, amely, CMF 1.3-as változat alatt a *CMF-1.3/CMFDefault* könyvtárra mutató közvetett hivatkozás, megtalálhatjuk a tartalomtípusokat meghatározó `File.py` és a `NewsItem.py` Python-modulokat. Ha kíváncsiak vagyunk, hogyan állíthatjuk be a tulajdonságok alapértékeit, nézzük meg a `factory_type_information` változót bármelyik CMF-típus akármelyik moduljában.
- *Product factory method*: megadja azt a tagfüggvényt, amelyet a CMF a típus új példányainak a létrehozásához hív majd meg.
- *Filter content types* és *Allowed content* types: ez a két bejegyzés ugyan két különálló tulajdonság, de együvé tartoznak. Mindkét tulajdonság valamennyi CMF-típusban megtalálható, kizárólag a mappa stílusú elemekre vonatkoznak. Ilyen például a *Folders* vagy a *Topics*. Az első, a `Filter content types` kétértékű változó, amely az `Allowed content types` érvényességét kapcsolja ki-be. A második, az `Allow content types` változó azt mutatja meg, hogy milyen altípusok kerülhetnek az adott típusba. Tehát például, ha egy olyan könyvtárat szeretnénk létrehozni, ami kizárólag hírelemeket tartalmazhat, ezt a *yes*-re való kattintással könnyedén megtehetjük, majd megmutatva, hogy mely típusokat engedélyezzük.

Új típus létrehozása

Új CMF-típusokat úgy hozhatunk létre a legkönnyebben, ha egy már létező típusra alapozva a webalapú CMF-típuskészítő eszközt használjuk. Ez a módszer ugyan nem engedi meg, hogy a típussal kapcsolatos mezőket vagy tagfüggvényeket megváltoztassuk, viszont lehetővé teszi, hogy szabadon állíthassuk be a típus műveleteivel kapcsolatos engedélyeket, megadjuk, hogy a típus megvitatható-e, sőt még az adattípus megjelenítésének a módját is megengedi.

Például lépjünk be a ***portal_types*** eszközbe, és a jobb felső sarokban található ***Select type to add...*** menüpontból válasszuk a gyárilag meglévő típusadatot. Itt két adatot kell majd megadnunk: az új típus nevét vagy azonosítóját, illetve azt a már létező típust, amelyre alapozni akarjuk. Az ***ATFDocument*** típust fogjuk létrehozni, amit értelemszerűen a ***Document*** nevű CMF alapértelmezett típusra fogunk alapozni.

Miután az új típust létrehoztuk, az összes típuslistában látható és elérhető lesz, ideértve a típuseszközöket és a tartalomnézetet, amelyben a típusból új példányokat készíthetünk. Tulajdonképpen bárki, aki rendszergazdai jogosultságokkal rendelkezik az oldalon, mostantól megtalálhatja az új ***ATFDocument*** típust abban a menüben, ahonnan az újonnan létrehozandó új típusokat is kiválaszthatja.

De mi értelme van ennek, ha egyszer az ***ATFDocument*** és a ***Document*** teljesen egyforma? Nos, valójában nem teljesen egyformák, inkább csak azonos tagfüggvényeken és általános osztálymeghatározásokon osztoznak. A típus egyéb jellemzői, mint például a tulajdonságok, az engedélyek és a bőrök (skins) alapértelmezés szerint azonosak ugyan a ***Document*** beállításaival, de bármikor teljesen megváltoztathatják a kinézetet. Így például, ha azt szeretnénk, hogy a ***Document*** fehér alapon feketével jelenjen meg, vitafórum nélkül, az ***ATFDocument*** viszont gesztenye alapon sárga színnel és vitafórummal, ezt ezzel a módszerrel minden további nélkül, néhány kattintással kialakíthatjuk. Azután, amikor a későbbiek során CMF-rendszerünket fejlesztjük, az ***ATFDocument*** típus a ***Document*** típussal együtt önműködően frissülni fog.

A színfalak mögött

Könnyen előfordulhat azonban, hogy olyan típust szeretnénk, amely a meglévő típusoktól teljesen eltérő mezőkkel vagy viselkedéssel bír. Erre is létezik néhány lehetőség, ezek közül a legrugalmasabb (s egyben legnagyobb kihívást jelentő, és legsilányabbul dokumentált) módszer a CMF-szabályoknak megfelelő új Zope-termék létrehozása. Minden Python-csomagnak a csomag gyökérkönyvtárában tartalmaznia kell egy ***__init__.py*** állományt. Ez az állomány akár üres is lehet, de itt helyezhetjük el azokat az utasításokat, amelyek a modul első memóriába töltése során elindulnak. A termékek esetében maga az osztály az `initialize()` tagfüggvény segítségével az ***__init__.py*** belsejében jegyzi be magát a Zope rendszerébe, ami egyetlen, általában `context`-nek nevezett értéket fogad. A lecsupaszított Zope-termék tehát egyetlen ***__init__.py*** állományból áll, ami a következő misztikus `MyProduct`-termékhez hasonlít:

```
import MyProduct
def initialize(context):
  context.registerClass(
    MyProduct.MyProduct,
    constructors=
    ➥(MyProduct.manage_addMyProductForm,
      MyProduct.manage_addMyProduct)
    )
```

A Zope induláskor végignézi a termékeket és a megfelelő összefüggésekkel (`contex`) meghívja az `initialize()` tagfüggvényeket. Az összefüggések a Zope szerzeményező rendszerének a részei, ahol az objektumok tulajdonságait a rendszerben elfoglalt helyük legalább annyira meghatározza, mint az eredeti osztálymeghatározás. A fenti példában a `MyProduct` két létrehozóval (constructor) jegyzi be magát: ez a `manage_addMyProductForm` és a `manage_addMyProduct`. A CMF-típusnak nemcsak a Zope, hanem a CMF rendszerébe is be kell jegyeznie magát, már amennyiben meg akar jelenni a különféle CMF-eszközök között. Termékünk `initialize()` tagfüggvényének ezért a CMF-hez tartozó bejegyzési műveleteket is tartalmaznia kell, azaz a ***__init__.py*** programnak modulokat kell importálnia a CMF-ből. Továbbá minden CMF-típusnak be kell jegyeznie magát a `Products.CMFCore.utils` egyik CMF vonatkozású alaphelyzetbe állító eljárásával. Például a CMF-be épített ***CMFDefault __init__.py*** programja első lépésként megadja azokat az osztályokat, amelyeket az elkövetkezendőkben be fog jegyezni:

```
contentClasses = ( Document.Document
                 , File.File
                 , Image.Image
                 , Link.Link
                 , Favorite.Favorite
                 , NewsItem.NewsItem
                 , SkinnedFolder.
                   ➥SkinnedFolder
)
```

Majd valamennyi osztályhoz megad egy létrehozót:

```
contentConstructors = ( Document.addDocument
     , File.addFile
     , Image.addImage
     , Link.addLink
     , Favorite.addFavorite
     , NewsItem.addNewsItem
     , SkinnedFolder.addSkinnedFolder
)
```

Természetesen minden osztály saját egyedi eszközzel rendelkezhet:

```
tools = ( DiscussionTool.DiscussionTool
        , MembershipTool.MembershipTool
        , RegistrationTool.RegistrationTool
        , PropertiesTool.PropertiesTool
        , URLTool.URLTool
        , MetadataTool.MetadataTool
        , SyndicationTool.SyndicationTool
)
```

Végül a csomag `initialize()` tagfüggvénye, amit egy kicsit lerövidítve adunk közre, eszközök esetében a `utils.ToolInit()`, tartalom esetében pedig a `ContentInit` eljárással bejegyzi az osztályokat. Ezután a visszakapott értékkel meghívja az `initialize(context)` függvényt, ezáltal jegyezve be az új terméket a Zope rendszerébe:

```
def initialize( context ):
   utils.ToolInit(´CMFDefault Tool´,
                  ➥tools=tools,
```

```
                    product_name='CMFDefault',
                    icon='tool.gif',
                    ).initialize( context )

utils.ContentInit( 'CMFDefault Content'
          , content_types=contentClasses
          , permission=AddPortalContent
          , extra_constructors=
            ➥contentConstructors
          , fti=Portal.
            ➥factory_type_information
          ).initialize( context )
context.registerClass(Portal.CMFSite,
      constructors=(
      ➥Portal.manage_addCMFSiteForm,
      ➥Portal.manage_addCMFSite,
                    ))
```

Láthatjuk, hogy az `initialize()` fenti változatának utolsó utasítása igen hasonló a `MyProduct()` példa `initialize()` függvényének utolsó részéhez, ami ékesen bizonyítja, hogy a CMF-típusok valójában néhány további kiegészítéssel rendelkező Zope-termékek.

Érdemes használni a CMF rendszert?

E cikk zárja a Zope mint tartalomkezelői felület körüli vizsgálódásainkat. Körutazásunk a Plone-nal kezdődött, és a CMF-rendszerrel, illetve a CMF-típusokkal zárult. Most, hogy már egy kicsit részletesebben is megismerhettük a CMF rendszerét, eldönthetjük, érdemes-e CMS alapú projektekbe belekezdenünk.

Jó hír, hogy CMF igen hatékony és rugalmas rendszer. Egy ügyes és hozzáértő fejlesztő kezében lehetővé teszi, hogy saját CMS rendszert hozzunk létre a piacon jelenleg elérhető üzleti rendszereknél jóval olcsóbban, ugyanakkor nagyobb rugalmassággal. Minthogy a rendszer teljes egészében a gyors fejlesztésekhez szánt Zope-ra épül, az új típusok létrehozása, a sablonok módosítása és az új szolgáltatások fejlesztése igen egyszerű és gyors.

Ugyanakkor a CMF, akárcsak a legtöbb nyílt forrású program, iszonyú hiányt szenved naprakész, használható leírások terén. Bizonyos vagyok benne, hogy a Plone CMS rendszerének sikere nagyrészben a Plone-hoz tartozó kiváló leírásnak köszönhető.

Tehát amennyiben használni szeretnénk a CMF-et, készüljünk fel egy komoly adag Python-kód átrágására, nem kevés kísérletezgetésre és más CMF-fejlesztők segítségének igénybevételére. Figyelembe véve a CMF központi szerepét a Zope világában, véleményem szerint a CMF-leírások minősége és mennyisége folyamatosan javulni fog. Amíg azonban ez az idő el nem jön, a CMF-fel való munka sok-sok türelmet, forráskódböngészést, próbálkozást és hibajavítást igényel.

A CMF rendszert jelenlegi állapotában – a legnagyobb és legösszetettebb tartalomkezelő rendszerek kivételével – egy kicsit haboznék bármi másra felhasználni.

Ahogy mondják, a CMF rugalmassága és hatékonysága pontosan ebben a méretarányban mutatkozhat meg a leginkább. Röviden: amennyire a CMF nem megfelelő a kisebb munkákhoz, valószínűleg épp annyira jól működik a nagyoknál. Az idő múlásával várhatóan egyre jelentősebb szerepet tölt majd be a nyílt forrású tartalomkezelésben, keretrendszert biztosítva egy saját CMS programrendszer gyors kifejlesztéséhez.

Összefoglalás

A Zope CMF saját CMS-készítéshez használható lenyűgöző keretrendszer. Kétségem sincs afelől, hogy a CMF alatt könnyedén lehet CMS rendszert készíteni, ráadásul lényegesen kevesebb költséggel és erőfeszítéssel, mint amennyit egy teljes értékű üzleti megoldás igényelne. Azt mondják, a CMF ideje még nem érkezett el azok számára, akik nem állnak bensőséges viszonyban vele vagy nem szeretnének jelentős időt pazarolni a megtanulására. Elgondolkodtató, hogy a CMF a Plone által került reflektorfénybe, az a tény pedig, hogy a Zope 3 közel vagy teljesen a CMF-fel egybeépítve jelenik majd meg, azt sugallja, hogy a Zope Corporation mostantól komoly késztetést érez a CMF lenyűgözőbbé és jobban dokumentálttá tételére, mint korábban.

Amennyiben komolyabb programozói tapasztalatokkal rendelkezünk a Python és a Zope területén, szinte biztosan fel tudjuk használni a CMF-rendszert saját típusaink Zope-termék alakú létrehozásához, velük pedig lenyűgöző és érdekes oldalakat készíthetünk magunknak és ügyfeleink számára. Ugyanakkor, amíg a típuskészítő rendszer nem lesz egy kicsit jobban érthető, a Zope-közösségen kívül a CMF nem fogja az őt megillető figyelmet megkapni. A Zope-termékek létrehozása ma már nem tűnik olyan fekete mágiának, mint korábban, és várhatóan a jövőben a CMF-típusok készítésére is ugyanilyen szemmel tekintünk majd.

A következő hónapban hirtelen fordulatot veszünk, és egy másik nyílt forrású CMS rendszert vizsgálunk meg: a Bricolage-t. A Bricolage, ami Mason-, `mod_perl`- és PostgreSQL-alapokra épül, igen szép területet hasított ki magának az elmúlt években, és egyre jelentősebb résztvevője a nyílt forrású CMS-közösségnek.

Linux Journal 2003. augusztus, 112. 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 Izraelben, Modi'in-ben él.

KAPCSOLÓDÓ CÍMEK

A Zope CMF honlapja a ➲ http://cmf.zope.org

Már a honlapon sem tudtam egyszerűen eligazodni, ráadásul semmilyen jó minőségű, hasznos útmutatót sem találtam CMF-témában.

A legjobb, CMF-típusokról szóló bemutatkozó jellegű írást nem a CMF-oldalon, hanem a Plone oldalán, a ➲ http://www.plone.org címen találtam. Például a ➲ http://plone.org/documentation/CMFTypesBook/backtalk_book_view címen fellelhetjük a CMF Types Book írást, amely jól olvasható és számos példát tartalmaz. A The Plone Book 8. fejezete szintén tartalmaz néhány hasznos útmutatót a CMF-típusokról. Ezt a ➲ http://plone.org/documentation/book/8 címen találjuk. Mint mindig, a ➲ http://www.zopelabs.com címen található a ZopeLabs szép számú példakódja és minibemutatója.

A hálózat rejtett zugai

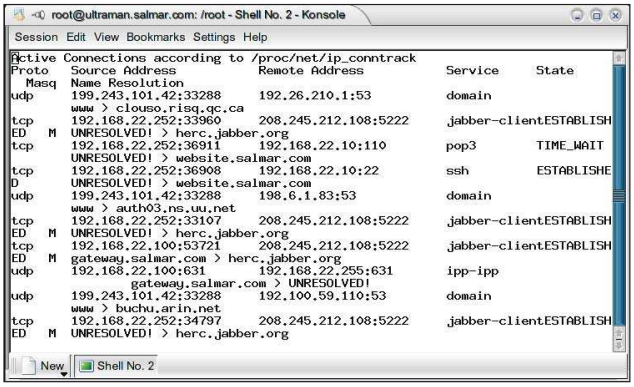
Hálózatunk legsötétebb sarkainak megvilágításához Marcel a hálózatmegfigyelő eszközökből állított össze egy különleges menüt.

Nem, François, ennek a szimatolónak semmi köze a borhoz. A bor az a terület, ahol az emberi orr bármilyen programnál jobban teljesít, függetlenül attól, hogy milyen okos a programozó. Őszintén szólva, mon ami, a borkóstolás – természetesen minőségellenőrzési célzattal – olyan feladat, amit nem szívesen automatizálnék. Azoknak a szimatolóknak, amikkel a Linuxszal való kotyvasztás közben találkozhatunk, más a rendeltetése.

Nézz csak ide, mon ami! Figyeld meg, hogy a sávszélességünk mekkora része van használatban itt és itt. Kíváncsi vagy, hogy ezek a kapcsolatok mekkora sávszélességeket jelentenek? François, miért nem ide figyelsz? Á, megérkeztek a vendégeink! Miért nem szóltál?

Bonsoir, mes amis! Örömmel üdvözöllek titeket ismét Chez Marcelnél, a kitűnő Linux-konyha, a világ legjobb borai és a nyílt forrású dolgok iránt érzett általános szeretet házában. Üljetek csak le, és helyezzétek magatokat kényelembe! Mielőtt megérkeztetek, éppen arról beszéltem François-nak, hogy mennyi rejtett adat áramlik keresztül egy átlagos hálózaton. Ha már a rejtett élvezeteknél tartunk, François, kérlek, siess a borospincébe, irány a nyugati szárny, és hozd fel az 1995-ös Rioja Imperial Gran Reservát. Ez a spanyol vörös egy tökéletes hálózatos bor, nemde?

Éppen azt ecseteltem hűséges pincéremnek, hogy egy átlagos hálózaton mennyi minden történik, és sokan teljesen megfeledkeznek azokról a kapcsolatokról, amiket korábban ők kezdeményeztek. A működő kapcsolataink ellenőrzésére szolgáló legegyszerűbb eszköz, a Netstat minden Linux-rendszercsomagban megtalálható. A `-a` és `-p` kapcsoló segítségével rendszerünk szinte összes nyitott hálózati kapcsolata (vagy kapuja) feltérképezhető, és azt is megtudhatjuk, hogy melyiket melyik program használja. Figyeljük meg, mi történik akkor, amikor a programot futtatom. Használni fogom a `-n` kapcsolót is, ami a Netstatot arra utasítja, hogy ne habozzon az IP-címeket szimbolikus címekké alakítani. Ez egy kicsit gyorsítja a program futását, mert így nem hajt végre névfeloldást. Az eredmény egy meglehetősen hosszú lista is lehet, ezért a kimenetet a `more`-ra irányítottam, lásd a *listánkon*. Á, François, megérkeztél a borral, nagyszerű. Kérlek, tölts a vendégeinknek! A lent látható lista nem teljes, de az általam kapott is hiányos. Ennek oka, hogy az álcázott (masqueraded) kapcsolatok IP-táblái a Netstat számára nem láthatóak; ez az



1. kép A conntrack figyelőprogram megjeleníti az álcázott kapcsolatokat

adat egy másik helyen található, mégpedig a */proc/net/ip_conntrack* fájlban. A PID a kapcsolatot használó program folyamatazonosítója. Éppenséggel kiadhatunk egy `cat` parancsot a */proc/net/ip_conntrack* fájlra vonatkozóan, de az eredmény nem lesz egy szemnyugtató olvasmány. Pillantsunk az alábbi példára (a kimenet egyetlen burkolt sor):

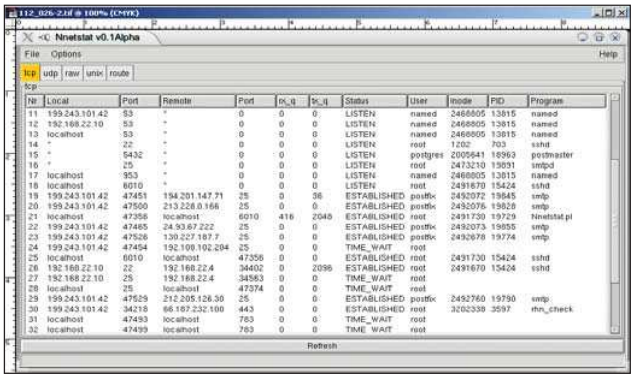
```
tcp       6 431253 ESTABLISHED
➥src=192.168.22.5 dst=192.168.22.10
➥sport=34212 dport=22 src=192.168.22.10
➥dst=192.168.22.5 sport=22
➥dport=34212 [ASSURED] use=1
```

Patrick Lagacé is nyilván nehezen olvashatónak találta ezt a szöveget. Az ő Conntrack-néző parancsfájlja a ➲ http://cv.intellos.net címen érhető el. Mivel egy Perl nyelvű parancsfájlról van szó, a letöltés után a jogosultságok megváltoztatásával egyszerűen tegyük futtathatóvá a fájlt, és adjuk ki a következő parancsot:

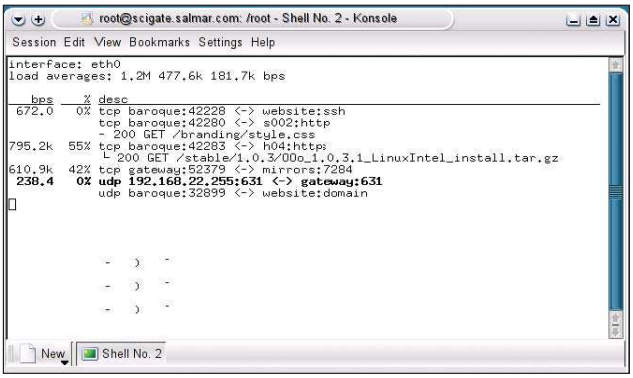
```
chmod +x conntrack-viewer.pl
./contrack-viewer.pl
```

A kimenet alapértelmezésben az összes kapcsolatot megmutatja, az álcázottakat is beleértve. A `-m` kapcsolóval korlátoz-

```
#netstat -apn | more
Active Internet connections (servers and estabilished)
Proto  Recv-Q  Send-Q   Local Adress        Foregin Adress       State         PID/Program name
Tcp         0      20   192.168.22.100:22   192.168.22.100:1014  ESTABILISHED  4003/sshd
Tcp         0       0   192.168.22.100:22   192.168.22.100:1015  ESTABILISHED  6122/named
Tcp         0       0   192.168.22.100:53   0.0.0.0:*            LIStEN        6122/named
Tcp         0       0   127.0.0.1:53        0.0.0.0:*            LIStEN        6122/named
Tcp         0       0   0.0.0.0:*           0.0.0.0:*            LIStEN        1231/httpd
Tcp         0       0   0.0.0.0:443         0.0.0.0:*            LIStEN
```



2. kép Az Nnetstat egy tetszetős grafikus Netstat



4. kép Az azonosított fájlnevek a pktstat megjelenítőjén

3. kép A Driftnet munkában: minden képetek hozzám tartozik!

hatjuk a kimenetet az álcázott kapcsolatokra, az ellenkező hatás (az álcázott kapcsolatok elrejtése) a `-d` kapcsolóval érhető el. Vessünk egy pillantást az *1. képre*, amelyen a program kimeneti képe látható.

Alexander Neptun Nnetstat nevű programja tetszetős külsejű grafikus segédprogram a működő hálózati kapcsolatok, útválasztó táblázatok és egyebek megjelenítésére. Saját legfrissebb példányunkat a ➲ http://www.aneptun.de/linux/Nnetstat címről tölthetjük le. A program alapjában véve egy Perl-parancsfájl, így telepítésre nincs is szükség, elég lehetőséget teremteni arra, hogy a ***Nnetstat.pl*** fájl futtatható legyen. Mint kiderül, a Nnetstat futtatásához szükség van még a ***Gtk.pm*** modulkönyvtárakra, és míg a Perl minden bizonnyal a rendszerünkön van már, ez a modul valószínűleg nincs. A beszerzés legegyszerűbb módja a Perl CPAN adattárról való letöltés, és a telepítő parancssor is egész barátságos:

```
perl -MCPAN -e "install Gtk"
```

Ha ez az első alkalom, amikor ilyen módon telepítünk Perl-modulokat, először keresztül kell jutnunk egy kis kérdés–felelet részen. Menjünk végig rajta, és fogadjuk el a felajánlott értékeket – bízzunk a rendszerben. Amit meg kell adnunk, az a legközelebbi CPAN-tükrözések címe. Amikor a rendszer felteszi a kérdést, válasszuk ki a földrészt és az országot, majd az elérhető helyi tükrözéseket. Ha ezzel készen vagyunk, a Gtk telepítése magától folytatódik.

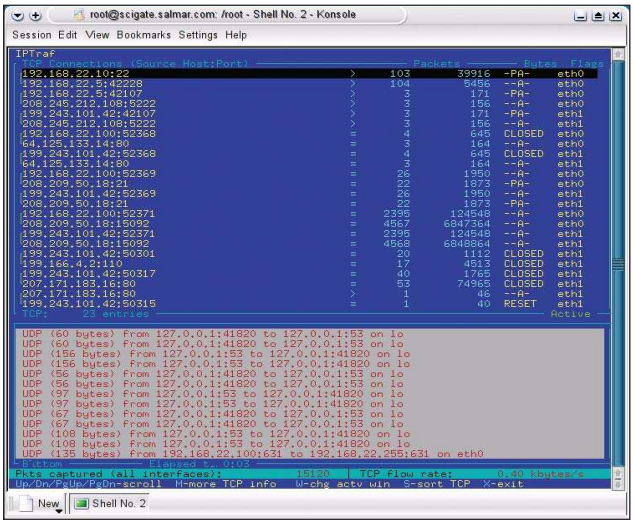
A Gtk Perl-modulok telepítése eltart egy darabig. Talán nem árt felhívnom a figyelmet, hogy a telepítés végefelé még egy tesztsorozat lefut. Ne lepődjünk meg, ha egy grafikus ablak jelenik meg a képernyőn, azt kérve, hogy a csomaggal kapcsolatos grafikus varázslatok kipróbálása céljából kattintsunk a ***Run*** feliratra. Amikor elégedettek vagyunk az eredménnyel, a próba befejezéséhez kattintsunk a ***Close*** gombra, és végezzük el a telepítésből hátralévő részt.

Ha igazán rémisztő – vagy szórakoztató, ez nézőpont kérdése – módon szeretnénk látni, hogy pontosan mi folyik keresztül a rendszerünkön, használjuk a Driftnetet. Már maga a név (vonóháló) elég ahhoz, hogy az embernek a hideg kezdjen futkosni a hátán. Röviden a Driftnet figyeli a kiválasztott csatolófelületen áthaladó kép- vagy videofájlokat (csak az MPEG típusúakat), és a talált képeket megjeleníti. Hogy ez a felfedés a rendszergazda számára félelmetesebb-e, hiszen kiderül számára, hogy a felhasználók miket néznek, esetleg maguknak a felhasználóknak, az több tényezőtől függ. A képgyűjtemény teljesen válogatás nélküli, semmilyen módon nem utal egy valóságos felhasználóra.

Saját példányunk beszerzéséhez keressük fel ***Chris Lightfoot*** honlapját a ➲ http://www.ex-parrot.com/~chris/driftnet címen, és töltsük le a forráskódot. Mielőtt a köztetek lévő szellemidézők megkérdeznék – amikor utoljára ellenőriztem, a honlap még nem szűnt meg és nem is költözött melegebb éghajlatra. A Driftnet lefordításához szükség van néhány programkönyvtárra, ezek közül a legjelentősebb a ***libungif***, a ***libjpeg*** és a ***libcap***. Ha még nem telepítettük őket, a hivatkozások a cikk végén, a ***Kapcsolódó címek*** között megtalálhatók, de először a rendszercsomag lemezein érdemes keresnünk. A csomag lefordítása ettől kezdve egy egyszerű tarcsomag-kibontás és a `make` futtatása a forráskód könyvtárában. Ezután már futtathatjuk is a kicsomagolt programot a könyvtárból, vagy átmásolhatjuk egy megfelelőbb helyre:

```
./driftnet -i eth0
```

Mivel a Driftnet futtatásához a csatolófelületet vegyes módra kell állítani, a futtatásához rendszergazdai jogosultságra van szükség. A *3. képen* a Driftnet működés közben látható. Kétségtelen, hogy a hálózatunkon keringő képek nézegetése jó szórakozás, ha nem törődünk a folyamat sávszélességigé-



5. kép Az IPTraf alapértelmezett megfigyelőablaka

nyével. De milyen egyéb érdekes dolgok mozognak ezekben a vezetékekben? Világhálós kérések, fájlletöltések, elektronikus levelek, üzenetváltások és még egy csomó más. A legtöbb hálózatfigyelő program – a Netstatot is beleértve – megmutatja a működő kapcsolatokat, de a következő kérdés az, hogy ezek pontosan mekkora forgalmat képviselnek.

David Leonard írt egy `ncurses`-alapú programot, amely a `pktstat` nevet viseli (➲ http://www.itee.uq.edu.au/~leonard/personal/software/#pktstat), és igen jó munkát végez az egyes kapcsolatok által lefoglalt sávszélesség bemutatásának terén. Az üzemben töltött idő formájában tárolja a mindenkori hálózatterhelési értéket, de nem a futtatási sor folyamatainak, hanem az átviteli sebesség nyomon követésével. A többi programtól az a képessége különbözteti meg, hogy a hálózatunkon lévő ügyfélgépekről letöltött vagy a webkiszolgálón áthaladó adatcsomagokhoz rendelt fájlneveket meg tudja jeleníteni. A pktstat fordítása a forráskód kicsomagolásából, a megfelelő könyvtárra való váltásból és a `make` parancs futtatásából áll:

```
tar -xzvf pktstat-1.7.2q.tar.gz
cd pktstat-1.7.2q
make
su -c "make install"
```

A program futtatásakor a `-i` kapcsolóval adhatjuk meg azt a csatolót, amelyiknek a forgalmát vizsgálni szeretnénk:

```
pktstat -i eth1
```

Ekkor egy, a *4. képen* láthatóhoz hasonló ablak jelenik meg. Mint látható, elkezdtem letölteni a legfrissebb Openoffice.org-csomagot. A tényleges fájlnév a kapcsolat adatai alatt jelenik meg, ugyanez érvényes a HTTP-webkérésekre. Nemcsak a letöltés alatt álló fájl címe látszik, hanem a fájl neve is, legyen akár egy HTML-oldal vagy egy kép.

Ha már a hálózati forgalomról esett szó, és ha egyszerűen arra akarjuk fáradozásainkat összpontosítani, hogy éppen hol és mire használják a hálózatunkat, ennek kiderítésére a mai menü legutolsó fogásaként felszolgált IPTraf a legmegfelelőbb. Szerény konyhafőnökötök egyik kedvenc IP-forgalomfigyelő segédprogramja, amihez időről időre mindig visszatér, az IPTraf. Ez egy `ncurses` alapú program, ami megjeleníti az IP-forgalmat, a csomagok és bájtok számát (beleértve a nem IP-csomagokat is), az UDP-forgalmat, a bejövő és kimenő adatok mennyiségét és még sok egyebet. Az IPTraf az a csomag, amelynek mindenkinél kéznél kell lennie, akit egy hálózat felügyeletével bíztak meg.

Látogassunk el ***Gerard Paul*** Java-honlapjára (➲ http://iptraf.seul.org), és töltsünk le egy IPTraf-példányt magunknak. Csomagoljuk ki a `tar` és `gzip` programokkal becsomagolt fájlt, lépjünk a könyvtárba, és a program lefordításához futtassuk a ***Setup***-ot. A telepítési folyamat a bináris állomány ***/usr/local/bin*** könyvtárba való másolásával fejeződik be. Az IPTraf futtatásához gépeljük be az `iptraf` parancsot, nyomjunk ENTER-t, és már sínen is vagyunk (az *5. kép* egy működő IPTraf-folyamatot szemléltet).

Miközben az IPTraf összegyűjti és megjeleníti az adatokat, a képernyő nagyon hamar megtelhet. Érdemes nagyobb, például 80×40 méretű X-terminálon futtatni a programot. Az ESC gomb megnyomásával visszatérhetünk a pillanatnyi nézetből vagy folyamatból. Innen megváltoztathatjuk a beállításokat, szűrőket adhatunk hozzá vagy távolíthatunk el, majd folytathatjuk az adatgyűjtést. Az IPTraf különböző nézeteket kínál többek között az alapértelmezett állomások közötti forgalomról, a csatoló forgalmi kimutatásának alap- és részletezett adatairól, egyéb fizikai statisztikákról és a csomagméret-hibákról. Ne tévesszen meg a program látszólagos egyszerűsége. Az IPTraf elég rugalmas ahhoz, hogy az IP-megfigyelés számos igényét kielégítse.

Nos, mes amis, a záróra rohamosan közeledik. Mialatt François újratölti a poharaitokat, én annak a reményemnek adok hangot, hogy amikor elmentek, pontos képpel fogtok rendelkezni arról, hogy mi történik a hálózataitokon. A jó rendszergazdák tudják, hogy mi folyik a hálózatukon, ám emellett azt is tudják, hogy mit nem szabad észrevenniük. Ezzel emelem poharamat rátok, mes amis. A votre santé!

Bon appétit!

Linux Journal 2003. augusztus, 112. 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 (jelenleg is egy könyvön dolgozik).

KAPCSOLÓDÓ CÍMEK

Conntrack nézőke ➲ http://cv.intellos.net
Driftnet ➲ http://www.ex-parrot.com/~chris/driftnet
IPTraf ➲ http://iptraf.seul.org
Libjpeg (Independent JPEG Group) ➲ http://www.ijg.org
Libpcap (csomagbefogó programkönyvtár) ➲ http://www.tcpdump.org
Nnetstat ➲ http://www.aneptun.de/linux/Nnetstat
Marcel borlapja ➲ http://www.marcelgagne.com/wine.html
Libungif ➲ http://prtr-13.ucsc.edu/~badger/software/libungif
Pktstat ➲ http://www.itee.uq.edu.au/~leonard/personal/software/#pktstat

Többfelületes programozás wxWindowszal

A jól bevált wxWindows a most készülő Chandler csoportmunkaprogram eszközkészlete.

A többfelületes fejlesztés az utóbbi években mindennapos kifejezéssé vált – Redmond bugyraiban éppúgy, mint a nyílt forrású kezdeményezések háza táján. Használata szinte mindig a Java, .NET, a V vagy a Qt körül forog, de ne feledkezzünk meg a wxWindowsról sem. Akár MFC-kódot akarunk Linuxra átültetni, akár egyszerűen csak a lehető legszélesebb közönség számára akarunk egy programot elérhetővé tenni, a wxWindows megállja a helyét. Furcsa módon úgy tűnik, hogy még nem sokan hallottak róla, de írásunk remélhetőleg segít abban, hogy szélesebb körben is megismerjék.

Miért választanánk a wxWindowst a fenti fejlesztőeszközök helyett? Jogos a kérdés, és minden egyes eszköz esetében más és más a válasz.

Összehasonlítás más eszközökkel

Bár a KDE-programok készítéséhez a Qt a szabványos eszköz, a wxWindows szintén használható. A Qt windowsos változata jogdíjas programok készítéséhez nem használható fel szabadon, de a wxWindows igen, és a Qt az eseménykezelő rendszerhez különleges előfeldolgozót igényel.

A Microsoft .NET és az Ximian Mono rendszere éppúgy, mint a Java, nem annyira eszközkészlet, inkább technológia. Ezek a megoldások újabb réteget adnak a programokhoz, ami a teljesítmény rovására mehet. Erről a kérdésről a „fejlesztési idő vagy futásidő” ellentét alapján is vitát lehetne nyitni, amit a program életciklusa befolyásol a leginkább. Ahelyett, hogy belebonyolódunk a kifejezések körüli csatározásokba, csak annyit mondunk: „mindenki másképp csinálja”.

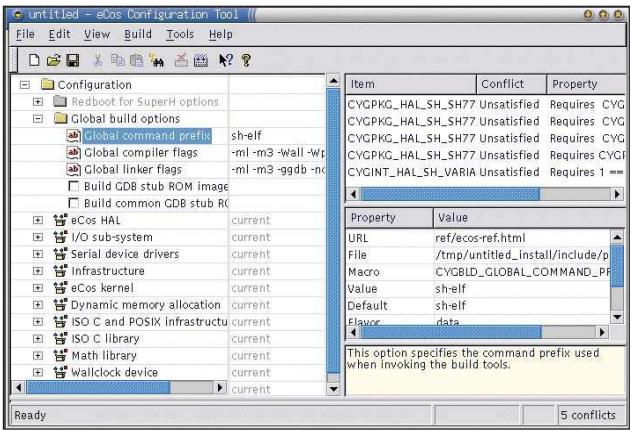
A V pedig egyszerűen nem olyan rugalmas, mint a wxWindows; nem támogat annyi fordítót. Még mielőtt a wxWindowst ezekkel a többfelületes fejlesztőeszközökkel igazán összevetnénk, fontos megemlítenünk, hogy a wxWindowsnak mindenféle összehasonlítás nélkül is megvannak a maga erősségei.

A wxWindowszal megkapjuk a nyílt forrás megszokott előnyeit – jogdíj fizetése nélkül felhasználható, szabadon módosítható, továbbá megszabadulunk attól a kockázattól, hogy a terjesztő cég megszűnik (vagy olyan módon frissíti a programkönyvtárait vagy a fejlesztőeszközeit, hogy kénytelenek legyünk megvenni az új változatot). Ezenkívül a programot a legkülönbözőbb operációs rendszereken is szabadon futtathatjuk, például Unix/Linux, Windows, Mac OS 9, Mac OS X és OS2 környezetben. Ráadásul minden rendszeren egységes megjelenést ad. A wxWindows felhasználási szerződése rugalmas, és a Nyílt Forrás Kezdeményezés (Open Source Initiative) is jóváhagyta.

Körülbelül 11 évi fejlesztéssel a háta mögött a wxWindows üzembiztos, kiforrott fejlesztői programkönyvtár, amihez a ➲ http://wxWindows.org honlapon elérhető levelezőlisták valós idejű támogatást nyújtanak. A több mint háromszáz osztály és ötezer függvény nagyszámú szolgáltatást tesz elérhetővé. Ezzel a könyvtárral az MFC programok átültetése is egyszerű feladat.

wxWindowst használó programok

Olyan vezető cégek is használják a wxWindowst, mint az AMD, az Intel Graphics Lab, a Compaq Alpha processzor fejlesztőcsoportja, a Netscape és a Lockheed Martin. Az Open Source Applications Foundation az új, Chandler névre keresztelt, személyes adatok kezelésére és csoportmunka támogatására szolgáló programjában használja (wxPython segítségével).



A Helpblocks (➲ http://www.helpblocks.com) programmal különböző felületekre készíthetők súgóállományok, és mind Windows-, mind Linux-rendszerre a wxWindows könyvtárral fejlesztik. Többek között wxWindowsos alkalmazás még a Vulcan 3D (modellezőprogram bányaipari felhasználásra, de még több is ennél), a SciTech Display Doctor programja, az Intuitive MX (egy többsávos hangkeverő, ami 3D-ben jeleníti meg a keverési folyamatot – az Intuitive Works fejleszti), valamint a Ground Control Station a Geneva Aerospace-től. Akár a te programod lehet a következő a listán!

Linuxos fejlesztés wxWindowszal

Szükségünk lesz a gcc-re, és olyan eszközöket használhatunk, mint a gdb, a ddd, az Emacs, valamint olyan egységes fejlesztőkörnyezetet, mint az Anjuta. Ha Windowsra is fordítunk, egy keresztfordítóra is szükség lehet. Legfőképpen pedig szükségünk lesz magára a ***wxWindows*** könyvtárra, ami szintén megtalálható a wxWindows-honlapon vagy megrendelhető CD-n is. A cikk írásának idején a legfrissebb változat a 2.4.0-s volt. (jelenleg a legfrissebb megbízható változat a 2.4.1-es, ami megtalálható az 51. CD Magazin/wxWindows könytárában). Sokunk jó szokásaival ellentétben nem árt elolvasni a csomagban található leírás linuxos fejlesztésre vonatkozó részét.

Nem árt feliratkozni a ***wx-user*** levelezési listára sem (a ➲ http://wxWindows.org lapon a ***Support*** alatt találjuk), hátha nem megy minden könnyűszerrel. Legokosabb, ha a kérdéseinkre rákeresünk az archívumban, hátha föltette már őket más is.

Maga a könyvtár több mint hetven példaprogramot tartalmaz,

Egy wxApp osztály létrehozása

```
#include "wx/wx.h"
#include "mondrian.xpm"

// Új alkalmazásosztály (wxApp)
// meghatározása
class MyApp : public wxApp {
    public:
    // kezdőérték-adás
    virtual bool OnInit();
};

// Új kerettípus meghatározása fő keretként
class MyFrame : public wxFrame {
    public:
    MyFrame(const wxString& title,
      const wxPoint& pos,
      const wxSize& size,
      long style = wxDEFAULT_FRAME_STYLE);

// Eseménykezelők (nem virtuális!)
    void OnQuit(wxCommandEvent& event);
    void OnAbout(wxCommandEvent& event);

    private:

    // Eseménytábla bevezetése
    DECLARE_EVENT_TABLE()
};

// Új alkalmazásobjektum létrehozása
IMPLEMENT_APP(MyApp)

// ´Main App´ megfelője
bool MyApp::OnInit()
{
    // Fő alkalmazásablak létrehozása
    MyFrame *frame =
    new MyFrame(_T("Example"),
                wxPoint(50, 50), wxSize(450,
340));
    frame->Show(TRUE);
    return TRUE;
}
```

amelyek szinte mindent bemutatnak, amire a könyvtárat valaha is eszünkbe juthat használni – valószínűleg ez a legjobb leírás, mert így működő programokkal kísérletezgetve figyelhetjük meg, hogyan is rakták őket össze. Ha a könyvtárat beszereztük, és a megszokott fejlesztőeszközeinkkel már kezelni tudjuk, kezdődhet a programozás.

Egy-két példa

Egyszerű alkalmazásosztályt létrehozni nem nehéz. *Listánkon* egy wxApp osztály létrehozására láthatunk példát. Amint látható, könnyen hozhatunk létre új wxWindows alkalmazást, és az MFC-t közelebbről ismerő olvasók talán elgondolkozva dörzsölik az állukat. Örömmel vehetik tudomásul, hogy nincs benne MFC, soha nem is lesz, de az osztályok egyszerűen átalakíthatók MFC-ből. Akad benne néhány dolog, ami az MFC-ben nincs, és nem támogatja az OLE-t.
Listánkon egy üres eseménykezelő is látható, ami olyan események kezelését teszi lehetővé, mint az egérkattintás, a billentyűleütések vagy a saját egyedi események. Ez a kód egy eléggé unalmas programot ír le, ami csak csücsül magában és meg se mukkan.
A könyvtár tele van működő példákkal – mintha minden osztályhoz lenne egy példadémon, ami bemutatja a működését. Az egyik az eseménykezelést bemutató példa, ami ***Vadim Zeitlin*** munkája. Azért választottam ezt, mert a legtöbb wxWindowszal ismerkedő fejlesztőnek gondja akad az eseményekkel – az olvasó tehát egy kis előnyre tehet szert. Kipróbálásához szükség lesz a ***wxWindows*** könyvtárra, és ha ez megvan, akkor a teljes forráskódja is megtalálható a ***Samples*** könyvtárban.
A honlapon még egy Wikire mutató hivatkozás is fellelhető, úgyhogy könnyen elérhető a legfrissebb leírás, és még bele is javíthat, akinek hozzáfűznivalója akad.

Adjuk meg a fejlesztőknek, ami jár!

A több ezer hozzájáruló mellett a fejlesztőcsapat magja külön említést és köszönetet érdemel:

- ***Julian Smart*** ➲ http://www.anthemion.co.uk
- ***Robin Dunn*** ➲ http://www.python.org
- ***Vadim Zeitlin*** ➲ http://www.tt-solutions.com
- ***Stefan Csomor*** ➲ http://www.advanced.ch
- ***Vaclav Slavik*** ➲ http://sourceforge.net/users/vaclavslavik
- ***Robert Roebling*** ➲ http://www.roebling.de

Ők a ***xw-user*** levelezőlistán gyakran napi, ha nem óránkénti rendszerességgel nyújtanak segítséget a wxWindows könyvtár felhasználóinak.

A wxWindows jövője

A 2.4.0-s változat kiadását követően a fejlesztőcsapat megadta azoknak a szolgáltatásoknak a listáját, amiket a 3.0-s változatban látni szeretnének. A támogatott felületek köre folyamatosan bővül, ez a következő kiadásban sem lesz másképp.
Az egyik új felület ***Marco Cavallini*** és Robert Roebling munkájának köszönhetően feltehetőleg a Windows CE lesz.
Készül a Winelib támogatás is a Winelib-csapat gondozásában, Julian Smart és Stefan Csomor pedig az MFC-ről wxWindowsra átültetést segítő jogdíjas eszközöket fejleszti.
Mivel egyre több cég használ Linuxot, MFC-kódjukat szeretnék Linuxra átültetni, és ennek egyik legegyszerűbb eszköze a ***wxWindows*** könyvtár. Már csak a kedvező költségekek miatt is megéri. Az átállással járó fejlesztési gondokat a könyvtár egyszerű felépítése a lehető legkevesebbre csökkenti, ezért a Windowsról Linuxra történő átállást tervező cégek kódjuk egyszerű átvitelére számíthatnak. Még wxNET felület fejlesztése is folyamatban van.
Lehet, hogy neked is van mit hozzátenned a wxWindowszal megvalósított programok hosszú sorához. A wxWindows programokról naprakész lista található a ➲ http://wxWindows.org honlapon.

Linux Journal 2003. július, 111. szám

Taran Rampersadt (➲ http://KnowProSE.com) 14 éves tapasztalattal rendelkező programfejlesztő; jelenleg tanácsadással és fejlesztéssel foglalkozik. Trinidad és Tobagoról ír. Részt vesz a helyi számítógépes társaságokban, és segít a Caribbean FLOS Konferencia (➲ http://floscaribbean.org) szervezésében.

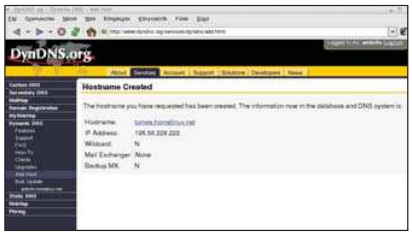
Webkiszolgáló az asztal alatt

Avagy hogyan készítsünk tartománynevet dinamikus IP-címmel rendelkező kiszolgálónknak.

A kimutatások szerint hazánkban az internet-hozzáférés még igencsak siralmas állapotban van, bár szép lassan javul a helyzet. Ez elsősorban a különböző DSL és kábeltévés hozzáférések terjedésének köszönhető. Ha egy ilyen csatlakozás végére egy linuxos kiszolgáló kerül tűzfallal, felmerülhet az igény, hogy egy kis webkiszolgálót is lehetne üzemeltetni. No nem azért, hogy hatalmas forgalmú csomópontot képezzünk információkban dúskáló portálok számára, hanem csak úgy, a saját szórakozásunkra, hogy kipróbálhassuk, hogyan is működik egy ilyen. Esetleg törzsügyfeleinknek szeretnénk zártkörű hozzáférést biztosítani az anyagainkhoz. Egy ilyen weboldal elérésével egyetlen gond akad, az, hogy nem tudjuk a címét, mivel az IP-cím időközönként változik.
Nézzük meg, hogy milyen megoldás lehetséges arra, hogy webkiszolgálónkat a változó IP-cím ellenére mindig állandó tartománynévvel érhessük el. A megoldáshoz egy ingyenes szolgáltatást fogunk igénybe venni.
A weben találhatóak olyan szolgáltatók, amelyek ezt a szolgáltatást kínálják, ilyen a DynDNS, a NoIp stb. Most a DynDNS-en keresztül mutatjuk be, hogyan is működik a dolog.
Első lépésként látogassunk el a DynDNS weboldalára. A ➲ http://www.dyndns.org oldalon megtaláljuk, amit keresünk. A nyitóoldalon válasszuk mindjárt a ***Services*** fület, és a 6. pontban már meg is találtuk azt, amire szükségünk van: a Dynamic DNS szolgáltatást.

A Dynamic DNS szolgáltatás

A dinamikus DNS-oldalra lépve tájékoztatást kapunk a szolgáltatás mibenlétéről, tehát itt kell majd beállítanunk a webkiszolgáló címét.
Mielőtt bármit tenni tudnánk, be kell jegyeztetnünk (registration) egy új hozzáférést, így a felhasználónevünkkel és jelszavunkkal a későbbiekben is módosítani tudjuk a beállításokat, ha az szükséges. A hozzáférést az ***Account*** menü alatt tudjuk bejegyezni. Itt el kell fogadnunk a felhasználási feltételeket, meg kell adnunk a felhasználónevet, elektronikus levélcímünket és jelszavunkat. Valós elektronikus levélcímet adjunk meg, mert a visszaigazolást erre a címre kapjuk meg – enélkül nem működik a szolgáltatás.
Amennyiben az ügyintézési részen túlestünk, jöhet az érdemi munka.



Most a frissen kapott hozzáférésünkkel már beléptünk, így a ***Dynamic DNS*** csoportban ki tudjuk választani az ***Add host*** menüt. A ***New Dynamic Host*** oldalon megadhatjuk a webkiszolgáló nevét, például sajatweb vagy homeweb.
A pont után egy szép nagy listából tartományt választhatunk a névhez – választhatjuk például a ***homelinux.net***-et, így a webkiszolgálónk címe: sajatweb sajatweb.homelinux.net lesz. Az IP-címnél a jelenlegi címünk látható, tehát a rendszer új tartománynevünket elsőként ehhez a címhez fogja rendelni.
Ezzel a tartománynév-bejegyzésen túl is estünk. Ha a böngészőnkben meghívjuk az új címet, máris a saját weblapjainkat láthatjuk a cím alatt. Igen ám, de holnap más IP-címünk lesz, ezért az oldalak nem lesznek elérhetők.
A feladat egy kis ügyfélprogrammal oldható meg. A ***Dynamic DNS*** csoportban a ***Clients*** menüpont alatt nagyon sok ilyen programot találhatunk, ízlésünknek és természetesen operációs rendszerünknek megfelelőt. Válasszuk ki a Unixot az operációs rendszerek listájából és a leválogatott listából a `ddclient` programot. Ez természetesen csak példa, lehet kísérletezgetni, hogy a többi hogyan működik. A `ddclient` telepítése nagyon egyszerű: kicsomagolás után be kell másolni a ***/usr/bin*** könyvtárba, majd a beállításfájlt a ***/etc*** alá:

```
cp ddclient /usr/bin
cp sample-etc_ddclient.conf
➥/etc/ddclient.conf
```

A másolások után már csak a beállításfájlt kell megfelelően finomhangolni. Hívjuk be kedvenc szövegszerkesztőnkbe, és szerkesszük át a példafájlt. Az alábbi sort keressük meg benne:

```
#use=if,      if=ppp+,
#via interfaces
```

Vegyük ki a megjegyzést az elejéről, majd pár sorral lejjebb a `login` és `password` sorokat már bejegyzett hozzáférésünk adataival töltsük ki. Most már csak meg kell adnunk, hogy melyik tartományt szeretnénk frissíteni. Pár sorral lejjebb, a `## dyndns.org dynamic addresses` csoportban az alábbi sorokat állítsuk be:

```
server=member.dyndns.org,
      protocol=dyndns2
      sajatweb.homelinux.net
```

Ha elindítjuk a `ddclient` programot és ezután változik meg az IP-címünk, a program a ***DynDNS*** kiszolgálóján frissíti a bejegyzést, és webkiszolgálónk a bejegyzett címen mindig elérhető lesz.

Ambrits Tamás (ambrits@ambrits.hu)
Nem „fanatikus” Linux-rajongó, de egyre több feladat megoldására alkalmazza nagy megelégedéssel.

Orbán Zsolt (orbix@mailbox.hu)
Igazi Linux-, de inkább Debian-megszállott. Minden érdekli, ami Linux.