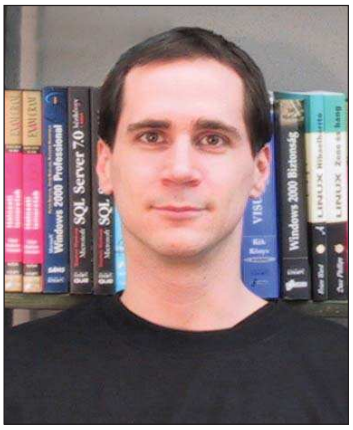
Egyre komolyabb erők mozdulnak meg a szakmai nyelvezet összehangolása érdekében. Nagy örömmel veszem például a ➲ http://hup.hu portálon belül kialakuló vitákat, az új lapszámokhoz mindig tartozik hozzászólás, és gyakran a nyelvi kérdések is előtérbe kerülnek. Ezúton is köszönöm a portál olvasóinak az értékes véleményeket!
Emellett talán a legizgalmasabb az új OpenOffice.org-magyarítás. A magyarítás a napokban készül el, sőt a használt szószedet is nyilvános! (Részletesebben lásd *Tímár András* írását a 13. oldalon.) Mint tudjuk, paradox módon elsősorban ezen a programcsomagon múlik majd az „irodai Linux” megítélése, hiszen az átlagos felhasználók többsége munkája túlnyomó részében ezt a programcsomagot használja.

access point (AP) → hozzáférési pont
átlátszó titkosított fájlrendszer (Transparent Cryptographic File System) → olyan fájlrendszer, amely beépített titkosítással rendelkezik. A titkosítás attól „átlátszó”, hogy a felhasználó számára nem érzékelhető (nem kell külön visszafejteni a fájlokat, ezt a feladatot a rendszer röptében elvégzi).
BID (business improvement district) → vállalkozásfejlesztési körzet
carrier grade → távközlési kategóriás (ipari távközlési célokra is használható).
chip → lapka
egykártyás gép → lásd: Single Board Computer
ESSI (Extended Service Set Identifier) → kiterjesztett szolgáltatásazonosító. Vezeték nélküli hálózatokban a szolgáltatások azonosítására használt szám.
fájlleíró (inode, i-node) → a fájlrendszerekben az egyes fájlokhoz tartozó adatok (például a fájl elhelyezkedése a lemezen, első részletének címe, tulajdonságai stb.) tárolására szolgáló adatszerkezet.
hot spot → (web)érzékeny terület, (Wi-Fi-) lelőhely
hook → horog, kapocs
hozzáférési pont (AP) → vezeték nélküli hálózatoknál azok az eszközök (vagy hálózatba kötött gépek), amelyeken keresztül csatlakozhatunk a vezeték nélküli hálózathoz.
hub → lásd: jelelosztó
hurokeszköz (lo) → a lo hálózati csatoló olyan cím, amelyen keresztül a saját gépünket érhetjük el a hálózati protokollon keresztül. Az IP-címek közül az összes 127.*.*.* cím ide tartozik. Elsődlegesen ellenőrzési célokra hozták létre, de több szolgáltatás is használhatja.
inode → fájlleíró
jelelosztó (hub) → hálózati eszköz, amely több csatlakozóval rendelkezik, és bármelyik csatlakozójára érkező hálózati forgalmat az összes többi csatlakozóra kiküldi.
lapkagép → olyan lapka, amely minden szükséges összetevőt tartalmaz ahhoz, hogy számítógépként működjön (cél-eszközökben használatos)
lo (mint hálózati csatoló) → lásd: hurokeszköz
SBC → lásd: Single Board Computer
scalable → méretezhető
Single Board Computer (SBC) → egy lapra szerelt (egykártyás) gép. Minden szükséges alkatrészt közvetlenül az alaplapra szerelnek. Leggyakrabban kis teljesítményű munkaállomásokhoz vagy célgépekhez használják. Lásd még: lapkagép
spinbutton → léptetőgomb
timer → időzítő, stopper
Transparent Cryptographic File System (TCFS) → átlátszó titkosított fájlrendszer.

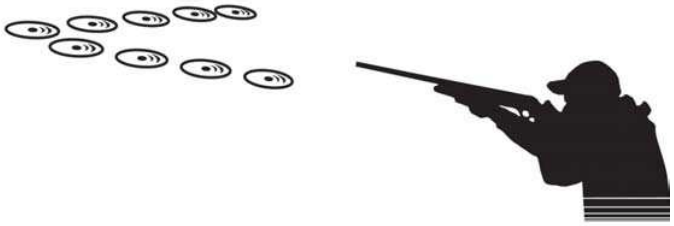
Beköszöntő



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

Időeltolódás márpedig van! Például New York és Budapest között olyan három év – legalábbis ami a számítástechnikát illeti. Sokak szerint ez kifejezetten előnyös számunkra. Nézzük például a pár éve történt dotnet-lufipukkantást: ha nem lenne meg köztünk ez a szerencsés három évnyi különbség, akkor a nagy durr idején már nálunk is rengeteg cég tönkrement volna egyik hétről a másikra. (Érdekes, az összeomlások viszont átlépnek ezen az időkorláton.)
Természetesen minden szinten igyekszünk ledolgozni ezt a három évet. Sok tekintetben azonban nagyon-nagyon hosszúnak tűnik majd ez az idő. Az egyik leginkább kézzelfogható (pontosabban a leginkább kézzel foghatatlan) terület a vezeték nélküli hálózatok világa. Csillogott a szemem, mikor Doc cikkét olvastam a New York-i WiFi-hálózatról (17. oldal), arra gondoltam, hogy a frissen szerzett laptopommal kiülök gyorsan kedvenc Orczy kertembe, kipróbálom, hátha valaki oda is telepített egy elérési pontot. Igaz, Tony sokkal jobb vadászeszközöket mutat be (Fedezzük fel a vezeték nélküli hálózatokat! – 24. oldal), de félő, hogy nálunk még egy katonai lokátornak sem vennénk túl nagy hasznát.
Akad viszont idehaza is izgalmas terv! Remélem, hogy az IHM által indított Közháló program (bővebben Anna ír róla a 12. oldalon) tényleg nagy sikerrel fejlődik majd, és végre ez a „hálózat" is megnyílik a nagyvilág számára. És ha már a minisztériumban is felvetődött a vezeték nélküli elérés gondolata, talán kevesebbet kell majd várnunk az átlagos időeltolódásnál.

Programvadászat



Devil-Linux

A mostani korongunkon található Devil-Linux egy tűzfalakhoz és átjárókhoz szánt Linux-kiadás. Ez egy meglehetősen kicsi (mindössze 50 MB-os), de testreszabható és biztonságos Linux. A telepítésével nem kell bajlódnia senkinek sem, a CD-ről egyből indítható a rendszer, ami egy tűzfalnál vagy átjárónál azért is jó dolog, mivel így nem kell attól tartanunk, hogyha a rendszerünket netán feltörik, akkor maradandó károsodást tudnak benne okozni. Ha a baj megtörtént, csak nyomjunk egy Resetet, és kész is van a javítás. Azonban ne feledkezzünk meg arról, hogy a behatoló által kihasznált rést azonnal be kell tömnünk, még mielőtt az internetre engednénk a gépet.

Nézzük a CD-ről indítható Linuxok, köztük a Devil-Linux előnyeit:

- A CD-n lévő fájlokat nem lehet módosítani, a behatoló lehet bármilyen ügyes, az esetlegesen telepített támadócsomag (root-kit) azonnal törlődik a memórialemezből (ramdisk), mihelyst újraindítjuk a gépet.
- A sok telepítéssel eltöltött idő helyett csak berakom, és már működik is a rendszer.
- Ha az egész rendszert frissíteni akarjuk, csak kicseréljük a CD-t, és kész is; rendszerindítás után már az új rendszer fog feléledni. Ez CD-írással és beállítással együtt nagyjából fél órát vesz igénybe.
- Biztonságos, nem kell többé merevlemez- és fájlrendszerhibáktól tartani.
- Áramszünet esetén sem történhet semmi rossz, egyszerűen csak újraindítás, és ismét él is a rendszerünk.
- Energiatakarékos, mivel nem szükséges merevlemezzel ellátni, ami egyébként sok energiát fogyaszt, tehát pénzt takaríthatunk meg, ráadásul így csendesebb is.
- Egyszerű: csak berakod a CD-t, és már működik is.

A hajlékonylemezes meghajtóra szükség van, hogy minden egyes újraindításnál a rendszer beállításait ne kézzel kelljen elvégeznünk. Mivel a CD-n levő fájlok nem módosíthatóak, a fejlesztők ehhez a következő megoldáshoz folyamodtak: a */etc* könyvtárat teljes egészében kitették.

A rendszer futtatásához szükséges követelmények:

- CD-ről történő rendszerindításra képes BIOS,
- hajlékonylemezes meghajtó a beállítások tárolására,
- minimum 486-os processzor és 64 MB memória.

A rendszer a 2.4.19-es rendszermagot használja `xfs`-sel. Támogatja az IP Tablest, ami a Firewall Builder által támgatott, így a szabályok gyártása viszonylag egyszerű lehet.

Programcsomagok

Nem minden alábbi csomagra van szükség, illetve nem mindegyik indul el önműködően, így mindenki eldöntheti, hogy mit szeretne futtatni a gépén.

- IPSec X.509-támogatással
- PPP Radius-támogatással a Portslave-en keresztül
- SSH
- DJBDNS
- BIND DNS-kiszolgáló
- STunnel
- PPPoE-támogatás, rendszermagmódú PPPoE
- Dinamikus DNS-szolgáltatások kezelése (NOIP és DynDNS)
- Cron
- DHCP-ügyfél
- DHCP-kiszolgáló
- PPTP-ügyfél
- PPTP-kiszolgáló
- Python v2, Python-LDAP-val
- LDAP-ügyfél és könyvtárai (OpenLDAP v1)
- Dynamic Routing-protokollok (BGP-4, RIP v1 & v2, OSPF)
- QoS és IPRoute2
- NTP-kiszolgáló és -ügyfél
- Perl
- SNMP
- Merevlemez-támogatás (Swap, Samba/Adat, Squid, Postfix)
- Betörésérzékelő rendszer, adatbázis-támogatással (Snort, MySQL)
- MySQL-ügyfél
- Samba-, Winbind- és ACL-támogatással
- XFS-, ACL-támogatással
- ReiserFS
- Chroot Vailek
- ADSL (PPTP-vel is)
- Squid HTTP proxy
- JFTPGW FTP proxy
- Postfix-, TLS-támogatással
- Friss Netfilter-kód, patch-o-matic „base” kiegészítéssel.

A fentebbi lista nem teljes, csak a fontosabb összetevőket soroltam fel. Amiket terveznek:

- Levelezés támogatása vírusszűréssel
- HTTP- és FTP-kiszolgáló
- Stealth Firewall
- (Stealth) proxyk (FTP, HTTP és minden más, amit még találnak)
- PostgreSQL-ügyfél.

A rendszer indítása

Kapcsoljuk be a gépet, tegyük a CD-t a meghajtóba, ha szükséges, a BIOS-unkat állítsuk át a CD-ről történő rendszerindításra. Helyezzük be a hajlékonylemezt a meghajtójába (ezt megelőzően az 52. CD ***devil-linux-0.5/etc.tar.gz*** fájlját rá kell másolni a FAT fájlrendszerre), és a gépünk máris elindul a Devil-Linuxszal. Biztosan lesznek hibák a rendszerindítás során, amiket viszont egyszerűen kijavíthatunk, ha a */etc* könyvtárban beállítjuk a rendszerünket.

Azoknak, akik ezt a Linuxot szeretnék használni, de nem akarják állandóan a Linuxvilág CD-t igénybe venni, iso formátumban is a feltettük a korongra – ebből különálló CD-ket lehet készíteni.

KDE

A Debian-felhasználók mindig hátrányban voltak a csicsás felületeket tekintve a többi kiadáshoz képes, ezért most közreadjuk a Debian/GNU Woodyhoz készített KDE 3.1.3-as csomagokat, hogy mindenki élvezhesse ennek a környezetnek a szolgáltatásait. Az `apt-cdrom add` paranccsal adhatjuk rendszergazdaként a meglévő ***sources.list*** fájlunkhoz.

Magazin

A magazinban szereplő cikkekhez tartozó anyagaink a szokásos Magazin könyvtárban találhatóak.

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

A jövő nyílt, de azért tőlünk vedd meg

Úgy tűnik, az IBM akkora üzletet lát a Linuxban, hogy tévéreklámban is érdemesnek látja népszerűsíteni. Az IBM – aligha egyedül – szeretne vezető szerepet játszani a linuxos világban, így a szakemberek szűkebb közösségén túllépve, a végfelhasználók felé nyitva minél szélesebb közönség előtt szeretné összekötni a saját nevét az operációs rendszerével. Az, hogy célját sikerül-e elérnie olyan versenytársak mellett, mint a szemfülesen „nyomuló” Lindows.com, erősen kétséges, ám ha a saját költségén teszi ismertebbé a Linux szót tévénézők százmilliói előtt, akkor kárt semmiképpen nem okoz.
A hirdetésben egy szőke, kék szemű, vagyis valószínűleg helyesnek, szeretni valónak szánt kisfiú testesíti meg a Linuxot. A némán ülő gyerek fejét híresség egész sora tömi tudnivalókkal, a legkülönfélébb gyakorlati és elméleti témakörökben. A kissrác minél több tudást szív magába, annál nagyobb hasznára lesz az emberiségnek – sugallja a film. A tudás megszerzése azonban csak az első lépés a bölcsesség felé, a közösség kialakításához meg is kell azt osztani – tanácsolja a fiúnak egy harvardi diák, ifjabb Gates, a linuxos közösség nyíltságára utalva. Család, tanulás, jövő, számítógép keveredik tehát a hirdetésben, amelynek üzenete nemcsak szerteágazó, de ugyanennyire zavaros is, így kérdéses, hogy hatására az átlagos amerikai tévénéző milyen szellemi mélységekbe merül le a focimeccsek szünetében. Igaz, legalább a záró jelmondat egyértelmű: „Linux. A jövő nyílt. IBM.” A többit akár el is hagyhatták volna.

Izmosodó Duronok

A felső kategóriás Opteron processzorok mellett a kisebb teljesítményű gépek piacáról sem fordította el a figyelmét az AMD. Új Duron lapkáit, bár az egész világon elérhetők lesznek, elsősorban a diplomatikusan fejlődő piacoknak nevezett Kelet-Európába és Kínába szánja. Az új processzorok 128 KB L1 és 64 KB L2 gyorsítótárat tartalmaznak, elődeiknél magasabb, 266 MHz-es FSB-vel üzemelnek, órajelük pedig 1,4, 1,6 és 1,8 GHz. Mivel nem állandó szorzóval kerülnek a boltokba, a régebbi, csak 200 MHz-es FSB-t támogató alaplapokban is használhatók lesznek, illetve – kellemes mellékhatásként – valószínűleg a túlhajtással kísérletezők körében is népszerűek lesznek.
➲ http://www.amd.com

Kigolyózva

Japán, Kína és Dél-Korea együttműködésének eredményeként olyan operációs rendszer jöhet létre, amelyik képes lesz felvenni a versenyt a Microsoft Windowsával – feltéve, hogy a témáról nyilatkozó japán kereskedelmi miniszter szavai valós szándékról árulkodnak, nem pedig külpolitikai–kereskedelmi célokat szolgálnak. ***Takeo Hirunama*** szerint az, hogy az elmúlt időszakban több vírus is viharos gyorsasággal terjedt el a Windows alapú rendszereken, kellő érv egy biztonságosabb operációs rendszer szükségessége mellett.
Tényleges tervek még nincsenek, ám az ötlet felvetése is jelezheti: a távol-keleti felhasználók, vásárlók türelme is véges. Amennyiben a munka valóban beindul, akkor az alapot valószínűleg a Linux és a különféle nyílt forrású fejlesztések jelentenék, végeredményként pedig megbízható és olcsó operációs rendszer jöhetne létre.
A kormányzati szervek részéről mindig fontos szempont az általuk használt alkalmazások forráskódjának nyíltsága, ez alapvető feltétel adataik biztonságának garantálásához. A Linux e téren mutatkozó előnyét a Microsoft azzal igyekszik csökkenteni, hogy szigorú feltételek mellett ugyan, de átadja a Windows forráskódját – az erről szóló szerződést nemrég írta alá éppen a kínai kormánnyal. Az óriáscég magas labdát kaphat, de biztosak lehetünk benne, hogy a válasz most sem marad el.

Bővülő Itanium-kínálat

Két új Itanium 2 processzort mutatott be az Intel, amelyek a jelenlegi Itanium lapkáknál kedvezőbb ár mellett is nagyobb teljesítményt ígérnek. Az 1 GHz-es órajellel futó, 1,5 MB gyorsítótárat tartalmazó Low Voltage Itanium 2 – mint a neve is utal rá – a megszokottnál alacsonyabb feszültséggel üzemel, így fogyasztása is csak 62 Watt körüli. A gyártó elsősorban sűrűn elhelyezett, állványra szerelt kiszolgálókba ajánlja, mivel ezeknél – a nehézkesebb hűtés miatt – különösen fontos a mérsékelt hőtermelés. A másik 64 bites jövevény az ugyancsak 1,5 MB gyorsítótárral rendelkező, 1,4 GHz-es Itanium 2, amely a néhány ezer dolláros árkategóriába eső kiszolgálókban, munkaállomásokban kaphat helyet. Az előbbi lapka ára 744, az utóbbié 1172 dollár.

➲ http://www.intel.com

Mindent látnak

A Linksys bemutatta WVC11B jelzésű internetkameráját, amely IEEE 802.11b szabvány szerinti, vezeték nélküli kapcsolaton keresztül képes továbbítani a képeket. A kamera saját IP-címmel rendelkezik, így teljesen önállóan tud kapcsolódni a meglévő ethernet vagy vezeték nélküli hálózatra, és MPEG-4 tömörítésű, 320×240 képpont felbontású mozgóképet szolgáltat. Képes a mozgásérzékelésre is, az eseményekről szükség esetén elektronikus levélben értesítést küld, illetve a képeket dátummal és időbélyeggel ellátva rögzíti. Segítségével könnyen figyelemmel tartható a ház egy része, a nyaraló vagy a telephely, és vezetékek lefektetésének költsége is megtakarítható. A kamera képét egyszerre négyen érhetik el, beállításai jelszóval védhetők. Újdonság, hogy a Linksys SoloLink néven dinamikus DNS-szolgáltatást is kínál kamerájához. Mivel az internetszolgáltatók általában minden csatlakozáskor más IP-címet adnak ügyfeleiknek, a kamera címe is gyakran változhat. Ennek követését segíti a SoloLink szolgáltatás, amelyet minden vásárló 90 napig ingyenesen próbálhat ki, utána húsz dollár éves díjért vehető igénybe. Az új kamera ára 230 dollár.

➲ http://www.linksys.com

Szuperkínai

Várhatóan már a jövő év márciusában elkészül Kína első szuperszámítógépe, ami másodpercenként tízbillió lebegőpontos utasítás végrehajtására lesz képes, és így joggal foglal el előkelő helyet a világ leggyorsabb számítógépeinek rangsorában. A 4000A jelzésű gépet a kiszolgálók gyártásával foglalkozó kínai Dawning cég építi. Az utasítások végrehajtását kétezer AMD Opteron 800 sorozatú processzorra bízzák, memóriából 2256 TB kerül beépítésre. A gépet időjárás-előrejelzésekhez, tudományos és olajkutatásokhoz fogják használni, ám ennél sokkal fontosabb üzenete van annak, hogy Kína egyedül, külföldi vállalatok bevonása nélkül is képes megépíteni egy ekkora szörnyeteget, és más nagyhatalmaktól függetlenül is el tudja végezni a kutatásaihoz szükséges számításokat.

Homo ludens

A Sony Ericsson telefonokhoz nemrég bemutatott kiegészítők elsősorban a játékosok figyelmét ragadhatják meg. A kicsit unalmassá váló „jaj de jó, hogy

mobilak vagyunk” gondolatkörön egy eredeti ötlettel túllépve a cég Gameboard EGB-10 jelzéssel egy játékvezérlőt, CAR-100 típusszámmal pedig egy apró játékautót kínál a vásárlóknak. Előbbi a Z600-as mobiltelefonnal használható, és valójában egy olyan nyolcirányú nyomógombos vezérlő, amelyet négy darab, szabadon választható szolgáltatással ellátott gomb egészít ki. Segítségével már-már játékkonzollá változtatható a színes kijelzős, polifonikus hangokat megszólaltató telefon, amelyre gyárilag a népszerű V-Rally autós játékot is telepítik. A CAR-100 azoknak a gyakorlatiasabb vevőknek készül, akik nem csak a kijelzőn szeretnék látni a versenyt. A gyufásdoboz méretű autócska Bluetooth-kapcsolaton keresztül, a telefon billentyűzetéről irányítható, és kiváló kiegészítőnek ígérkezik arra az esetre, ha a munkatársak vagy a járókelők várható átlagéletkorát szeretnénk statisztikailag kimutatható mértékben csökkenteni. A korlátozott darabszámban ez év vége felé megjelenő kisautókat főleg a gyűjtőknek szánják.

➲ http://www.sonyericsson.com

Gépelj bármit, megmondom, ki vagy!

A felhasználók gépelési gyorsaság és ritmus alapján végzett azonosítása nem új ötlet, ám általában nem foglalkoznak vele komolyan, hiszen a módszer egyrészt rengeteg tényezőtől függő mutatókra támaszkodik, másrészt megfelelő erősségre törekedve elég furcsa lenne jelszó helyett többsoros szöveget begépeltetni a számítógép elé leülő személlyel, mielőtt megkezdhetné annak használatát. Kiegészítő jelleggel, üzem közben, folyamatosan figyelve ugyanakkor alkalmas lehet annak észlelésére, ha a már azonosított felhasználó helyét más foglalta el. Az OS X alá készült BHand is hasonló célokra alkalmazható, segítségével például igazolható, hogy egy elektronikus levelet nagy valószínűséggel a feladóként megadott személy írt-e. Az egyelőre próbaváltozatként letölthető program készítői idővel biztonsági minősítést is ígérnek alkalmazásukhoz, addig is mint érdekességet érdemes lehet megismerni.

➲ http://www.10191.com/inferno/BHand/index.html

Linuxos Motorola telefon

Lassan megmutatkoznak a Motorola linuxos terveinek eredményei: elkészült a cég első, Linux-rendszerrel futó mobiltelefon-zsebtitkár készüléke. A már korábban bejelentett, felső kategóriás intelligens telefon eleinte csak a Távol-Keleten lesz elérhető, később Európában és Amerikában is megvásárolhatjuk.

A telefon – mint az ezen a piaci területen elvárható – nemcsak hangátvitelre alkalmas, de személyi adatkezelő alkalmazásokat is találunk benne, továbbá beépített kamerával rendelkezik, játékra, csevegésre, elektronikus levelezésre, animációk és MP3-zenék lejátszására egyaránt használható. Ára egyelőre ismeretlen.

A Motorola idővel minden telefonjában – az olcsóbb típusokban is – Linuxot szeretne látni. Ezzel egyelőre egyedül áll a jelentősebb gyártók között, és nem lesz könnyű dolga, ha a Symbian, Microsoft és PalmSource rendszerek kavalkádjában sikerre akarja vinni az elgondolásait. Az IDC becslései szerint hiába az olcsó és gyors alkalmazásfejlesztés lehetősége és a közösség támogatása, 2006-ra a Linux mindössze 4,2 százalékos részesedést tud majd szerezni a mobiltelefonok piacán. Valószínű ugyanakkor, hogy emiatt a Motorolánál nem keseredtek el túlságosan, hiszen a mobiltelefonok értékesítésén túl a Linuxot a beágyazott készülékek széles körében lehet használni.

➲ http://www.motorola.com

Melyik programot kéred, kis gazdám?

MXI OS néven egy korábban már felbukkant ötletbe próbál újra életet lehelni a Radixs. A különleges operációs rendszer feladata az, hogy teljes értékű munkaasztallá változtassa a zsebtitkárokat. A vezeték nélküli kapcsolattal ellátott hordozható eszközökön egy ügyfélprogramot kell majd telepíteni, amelyik a megjelenítést fogja végezni, az alkalmazások futtatásáról – amelyek Linux, Windows és Palm OS alapúak egyaránt lehetnek – pedig a mobilszolgáltató kiszolgálója gondoskodik majd. A felhasználó megfelelő előfizetés birtokában az asztali számítógépén megszokott alkalmazásokat is futtathatja majd mobil készülékén, feltéve, hogy meg tud békélni a kis felbontású kijelző és a körülményesen használható beviteli módszerek korlátaival.

➲ http://www.radixs.com

Újabb zseb-Linux

Szeptemberre teszik a legújabb linuxos zsebtitkár, a Softfield MX-7 megjelenését. A cég nem teljesen új játékos ezen a területen, mivel az Agenda Computingtől vette át az Agenda VR3 készülékek támogatásának terhét, most pedig a Sharp Zaurusának nyomdokain járva egy Qtopia-környezettel telepített saját zsebtitkárral próbálja meghódítani a rajongók szívét. Az MX-7 azon kívül, hogy Linuxot futtat, túl sok újdonságot nem kínál. Belsejében egy 200 MHz órajelű Motorola processzort találunk, 320×240 képpont felbontású kijelzője 65 536 szín megjelenítésére képes, 32 MB flash és 64 MB RAM memóriával rendelkezik, kezeli az SD és az MMC kártyákat, USB 1.1, RS-232 és PS/2 szabvány szerinti csatlakozóval, valamint beépített hangszóróval és mikrofonnal látták el. Operációs rendszere a 2.4.18-as rendszermagra alapul, a Zaurusra készült személyi adatkezelő programokat pedig várhatóan gond nélkül képes lesz futtatni. Ára 299 dollár.

➲ http://www.softfield.com

Helymeghatározás házon belül

A Hitachi új helyzetmeghatározó rendszere a vezeték nélküli számítógépes hálózat segítségével képes tárgyak, személyek helyzetének megál-

lapítására. Hasonló jellegű rendszerek már léteznek, gondoljunk csak a GPS-eszközökre, ám ezek viszonylag pontatlanok, vagy csak kisebb területek lefedésére alkalmasak, míg a Hitachi rendszere 1–3 méteres pontossággal dolgozik, és a 100–200 méterenként elhelyezett hozzáférési pontokkal viszonylag nagy területek lefedésére is használható. Működéséhez a keresett eszközt legalább három hozzáférési pontnak látnia kell, a helymeghatározás egyébként a szokásos módon, a rádiójelek válaszidejének mérése alapján történik. A WLAN alapú megoldás egyebek mellett gyárakban, raktárakban a munkadarabok mozgásának, helyének meghatározására, kiterjedt parkokban, intézményekben a látogatók útba igazítására vagy nagyobb irodákban a legközelebbi adott jellegű eszköz felkutatására használható.

Nem csak perelni tudnak

Két terméket is bejelentett a SCO: a UnixWare Office Mail Server 2.0-s és a SCO Authentication for Microsoft Active Directory 2.1-es változatát.

A UnixWare Office Mail Server kis- és közepes méretű vállalkozásoknak készül, a legújabb változata nemcsak az Eudora és a Netscape levelezők használatát segíti, hanem jelentősen továbbfejlesztett Outlook-támogatással is rendelkezik; az Office Mail Connector for Microsoft Outlook legújabb változatával használhatók a hagyományosnak mondható Outlook-szolgáltatások, mint a levelek megosztása, a közös naptárak, közös feladatkezelés, címtárak és vitacsoportok. A kiszolgáló támogatja a POP3 és az IMAP protokollokat, illetve LDAP alapú internetes címtár fenntartására is használható.

A SCO Authentication for Microsoft Active Directory használatával a Windows és a Unix alapú gépek békíthetők össze egymással. Segítségével központilag kezelhetők a Windows és Unix alapú gépeken található felhasználói fiókok, a hitelesítés pedig Kerberos protokollon keresztül végezhető. A SCO Authentication for Microsoft Active Directory a SCO saját rendszerei mellett támogatja a Windows Server 2003-at is, meghibásodás esetén önműködő feladatátadást tud végezni, többtartományos környezetben is eligazodik, valamint képes a Unix és Windows alapú gépek közötti idő összehangolására.

➲ http://www.sco.com/products/authentication

➲ http://www.sco.com/products/SCOoffice/mail

Replikáció PostgreSQL kiszolgálókhoz

A PostgreSQL Inc. bejelentette, hogy szabadon elérhetővé teszi kereskedelmi fejlesztésű eRServer 1.0 replikációs kiszolgálóját. A cég két évvel a kereskedelmi megjelenés után minden termékét szabadon is hozzáférhetővé teszi, ez a lépése is ennek a törekvésnek a része. A replikáció az egyik alapvető szolgáltatás a nagyvállalati szintű adatbázis-kezelő rendszereknél, ám a PostgreSQL eddig nem kínált ilyen lehetőséget, és így nem vehette fel a versenyt a kereskedelmi megoldásokkal. Az ingyenes termékhez támogatás alapesetben nem jár, de a fejlesztők reményei szerint telepítésével és felügyeletével kapcsolatosan egyre többen szereznek majd szerteágazó tapasztalatokat, így idővel a közösségi fórumokon is meg lehet majd találni az általános hibák elhárításának leírását.

➲ http://www.postgresql.com

Gyémánt alapokon

A japán NTT távközlési cég kutatói a németországi Ulm egyetemének munkatársaival együttműködve gyémánt alapú, 81 GHz-es órajellel működő félvezető eszközt készítettek. A gyémánt kedvező tulajdonságainak köszönhetően a jövő félvezetőinek egyik fontos alapanyaga lehet, elsősorban nagyfrekvenciás, nagy energiával üzemelő készülékekben, gigahertzes tartományban üzemelő erősítőkben, például digitális tévéadókban találkozhatunk majd vele. Gyémánt alapú eszközökkel már korábban is kísérleteztek, ám a kutatásokat hátráltatta, hogy esetükben a szennyeződések és az anyaghibák nagyságrendekkel rontják a teljesítményt. Japánban most sikerült ezeket a gondokat elhárítani és nagy tisztaságú, hibátlan kristályt növeszteni. A további fejlesztések során a kristály további finomításával szeretnék a működési frekvenciát 200 GHz-re, a kimeneti energiát pedig 30 W/mm értékre növelni, amellyel már a gyakorlati alkalmazás lehetősége is elérhetővé válna.

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

Rendszermagfejlesztési hírek

Az ingyenes programok és a nyílt forráskód világában a DRM szörnyű fenyegetésnek számít.

A rendszermagfejlesztés a 2.6-os változat felé araszolgat. A 2002. október 31-i „változatfagyasztás”, azaz a szolgáltatásbővítés befagyasztása nem sokáig tartotta magát, és a rendszermag azóta számos új tulajdonsággal bővült. *Linus Torvalds* és mások azonban szigorúbban kezelik, hogy mennyire szabálysértők az új fejlesztések. 2003 májusában Linus azt mondta, hogy a továbbiakban nem fogad el olyan foltokat, amelyek nem teljesen egyértelműek, hacsak a vezetőség más tagjai nem látták őket. Szerintem kevés esély van a 2.6-os rendszermag megjelenésére 2003 vége előtt, de az is elképzelhető, hogy a kódot 2003 szeptember vagy októbere körül valóban befagyasztják.

Linus Torvalds világosan leszögezte, hogy a Digital Rights Managementet (DRM) támogató foltokat elfogad. Az ingyenes programok és a nyílt forráskód világában a DRM szörnyű fenyegetésnek számít, lehetséges, hogy ez az ingyenes és a nyílt forrású programok végét jelenti. Linus úgy tartja, hogy a DRM-mel kapcsolatos alapvető rendszermagjellemzők önmagukban nem rosszak, de általános biztonsági gondokat okozhatnak. Már javasoltak erre néhány foltot, és ezeknek a rendszermaghoz történő igazításával Linus a szabad program jogellenes felhasználását akarja megakadályozni.

Roman Zippel új életet lehelt az HFS+ fájlrendszerbe, a Mac OS X rendszerek fájlrendszerébe. Az Ardis Technologiesnél dolgozó Zippel számos újítást vitt véghez egy létező, *Brad Boyer* készített HFS+ meghajtón – ideértve az írás és olvasás teljes támogatottságát, a jobb teljesítményt és a jobb közvetlen hivatkozásokat. Ez nagy könnyebbséget jelent az iPod- és más Mac-felhasználók számára, de sajnos – ahogy ez gyakran megesik –, Roman elfeledte elmondani Bradnek, hogy min dolgozik, így apró összetűzés támadt a fejlesztés jogain, miután Roman bejelentette az új kódot. Jelen cikk írásakor még nem biztos, hogy Roman vagy Brad vezeti-e majd a fejlesztését, esetleg más történik.

A National Security Agency (NSA) végrehajtott néhány API-változtatást saját SE Linux-változataiban, azért, hogy a 2.6-os megjelenésére SE Linux-foltok legyenek a 2.5 main tree-ben. Változtatásaik között szerepel az `ext3`-as fájlrendszer kiterjesztett jellemzőinek támogatása, ezzel gyakorlatilag bármiféle metaadat tárolható egy fájlban. A kiterjesztett tulajdonságokat (Extended Attributes – EAs) biztonsági rendszerek futtatására lehet használni, ilyen például a hozzáférés lista (Access Control Lists – ACLs) és a fájlrendszer-engedélyek. Az SE Linux mögött álló Linux biztonsági modul (Linux Security Module – LSM), szintén magára vonta az NSA figyelmét. Megváltoztatták a modulok kapcsait (module hook), hogy megakadályozzák a biztonsági rendszer sérülését egy fájl biztonsági címkéjének megváltoztatása és az ehhez kapcsolódó fájlleíró biztonsági mezők (inode security field) közötti (egy másodperc törtrészét képező) időben. Úgy tűnik, hogy a vezeték nélküli LAN lapkák (Wireless LAN chips) többek között a Broadcom BCM4306-es és a BCM2050-es a hullámhosszokon katonai kódok fogadására és továbbítására képesek. Ez lelassítja az ehhez a lapkákhoz készülő ingyenes meghajtók kialakítását, mivel az eszközgyártók a világ minden tájáról magukra vonják a különböző állami szervezetek haragját.

Zack Brown

Linux Journal 2003. szeptember, 113. szám

Ők mondták

Mindenki azt mondja, hogy „á, erre nincs gazdasági modell”, de igenis van: a cserepes virág gazdasági modellje… Cserepes virágokat rakok ki, hogy növeljem az önbecsülésemet és hogy szebbé varázsoljam a házamat. Ha majd' mindenki így tesz, akkor az egész város csodaszép lesz. Ugyanez igaz a kapcsolattartásra is. A telekommunikáció számára az a kihívás, hogy a vezeték nélkülit az igencsak intelligens készülékekkel ötvözze... Az eredmény sokszínű alkalmazást tesz majd lehetővé, a korábbiaknál hatékonyabb és szervesebb módon. *(Nicolas Negroponte, Nordic Wireless Watch)*

➲ http://www.nordicwirelesswatch.com/wireless/story.html?story_id=3152

Az információs technológia jövőjéről beszélgettünk, többek között a programbiztonsággal kapcsolatos kérdésekről… Tartottam magam ahhoz, hogy nyílt forráskódokat kell keresnünk, hogy ezeket könnyebben átvihessük a felhasználó készítette biztonsági algoritmusokra. A párbeszéd egyre bonyolultabb lett, mivel a nézeteink eltérőek voltak… Az a legroszszabb, hogy India a mai napig hisz a jogvédett termékek nyújtotta megoldásokban… Az IT-nek az egyén mindennapi életére kiható terjedése lesújtó hatást gyakorolna a társadalomra, ha az üzleti gyakorlatban a jogvédett termékekkel kapcsolatban akárcsak a legkisebb elmozdulás is megtörténne. Pontosan emiatt kell nyílt forrású programokat készíteni, amelyek az egész társadalom számára költségkímélők lennének. A nyílt forrású programoknak be kell törniük és meg kell ragadniuk Indiában, hogy az ott élő többmilliárd embernek a hasznára válhassanak. *(A. P. J. Abdul Kalam, India elnöke, ZDNet News/India)*

➲ http://news.zdnet.co.uk/story/0,,t272-s2135401,00.html

Itt nincs puccos felszerelés – a tesztelés során feltört fejlesztői táblát, villogó LED-eket használunk – és alkalmanként megszólal egy hangos csengő is, ami halálra rémít mindenkit. *(Andrew Greenberg a Portland State Aerospace Societyből a – Portland Állami Űrkutatási Társaság –, ahol Linuxot és más nyílt forrású programokat használnak a légkörön belüli műholdak fellövésekor)*

Linux Journal 2003, 113. szám

A világot figyelve

A Linux Journal új kiadvánnyal bővítette arzenálját, a Világnézővel (WorldWatch; ➲ http://WorldWatch.LinuxGazette.com). „A Linux az egész Földet behálózó jelenség, szerteágazó fejlesztése különféle társadalmi körülmények között zajlik. Ezért örömmel lépjük át cikkeinkkel, beszámolóinkkal és elemzéseinkkel a nemzetek és kultúrák közötti határokat"– mondta *Willy Smith*, az új kiadvány főszerkesztője. A Világnéző naprakész lapfigyelővel jelentkezik, amelyben a világ minden tájáról közölnek cikkeket a Linuxszal és a nyílt forrású programokkal kapcsolatban. A témakörök általában nem technikai jellegűek, a Linux fejlődését és a különböző közösségekre gyakorolt hatását mutatják be többféle szemszögből. A kifinomult visszajelzési rendszernek köszönhetően az olvasók hozzászólhatnak, beszélgethetnek és maguk is írhatnak az egyes témakörökről.

Doc Searls

Linuxos suligépek

Kaliforniában az iskolák és nonprofit szervezetek Mandrake Linuxszal futó ingyenes számítógépeket igényelhetnek az Amerikai Technológiai Képzési Alapítványtól (Technology Training Foundation of America – TTFA). A TTFA 1998 óta fogad adományozott számítógépeket a kaliforniai cégektől, és eddig több mint negyvenezer diák számára biztosított számítógép-hozzáférést. Számos cég asztali gépeinek 20–30 százalékát évente lecseréli, és a TTFA lelkesen várja ezeket a 2–4 éves számítógépeket. A TTFA irányítja a számítógépek szállítását a 16 számítógép-felújító partner telephelye felé. A gépeket felújító partnerek minden adatot eltávolítanak a gépek merevlemezeiről, szükség esetén megjavítják vagy kicserélik az egyes alkatrészeket, és a fogadó fél igényeinek megfelelően állítják be a számítógépeket.
A CDC és a TTFA háromféle operációs rendszert is kínál, ezek között megtaláljuk például a Mandrake Linux 9.0-t az OpenOffice.org-gal, a Gimppel és más csomagokkal együtt.
(Ehhez hasonló kezdeményezésekre Magyarországon is igen nagy szükség volna – a szerkesztő).
➲ http://www.computers2learnby.org

Walt Pennington

Linux-index

1. Az olyan windowsos asztali gépek száma a londoni Newham kerületben, amelyeket Linuxra és más nyílt forrású alkalmazásokra cserélnek le: 5000
2. Az olyan windowsos asztali gépek száma, amelyekkel ugyanezt tervezik, ha Nottingham követi Newham példáját: várhatóan 6500
3. Várható költségmegtakarítás mindkét esetben: egyharmad
4. A Linuxra átálló számítógépek száma Münchenben: 14 000
5. A Linuxra átálló rendőrségi számítógépek száma Alsó-Szászországban: 11 000
6. A Nyugat-Európában eladott olyan számítógépek mennyisége százalékos megoszlásban, amelyeken Linux található: 15
7. A széles sávú internet-hozzáférés százalékos növekedése 13 hónap alatt Európában, 2003 áprilisáig: 136
8. Ugyanezen százalékos adat az Egyesült Királyságban: 235
9. Az Európában széles sávú internet-hozzáféréssel rendelkezők mennyisége százalékos megoszlásban: 28
10. Az Egyesült Államokban széles sávú internet-hozzáféréssel rendelkezők mennyisége százalékos megoszlásban: 35
11. A Hongkongban széles sávú internet-hozzáféréssel rendelkezők mennyisége százalékos megoszlásban: 85
12. Ennyi millió európai fog széles sávú internetet használni: 2004 márciusáig: 50
13. Az olyan ausztrál McDonald's éttermek százaléka, ahol Wi-Fi található: 100
14. Azoknak a vidéki közösségeknek a száma a Loni-Shirdi térségben, a nyugati Maharashtrában, ahol pénzt gyűjtöttek egy műszaki körzet létrehozására: 200
15. A Wi-Fi lelőhelyek (hot spot) száma, amelyeket a közösség 2003 májusáig hozott létre: 50
16. Egy indiai szolgáltató 350 ezer km-es üvegszálvezetékének kihasználatlanul hagyott százaléka: 60–70
17. 2006-ig értékesítendő Wi-Fi-készülékek száma milliárdban: 2,3
18. A Cometa ennyi millió Wi-Fi-felhaszálót vár 2008-ig: 50
19. A Cometa ennyi millió Wi-Fi-készüléket vár 2008-ig: 100
20. A Cometa ennyi lelőhelyet (hot spotot) akar telepíteni 2004 februárjáig: 5000
21. A Cometa ennyi lelőhelyet (hot spotot) akar telepíteni 2005-ig: 20 000

Forrás
1–6.: ZDNet UK
7–12.: Nielsen/NetRatings
13.: New York Times
14–16.: The Hindu
17.: Jupiter Research
18–21.: Reuters

Linux Journal 2003. szeptember, 113. szám

Behálózva

A most indított közbeszerzésekre az IHM és az OM három év alatt várhatóan többmilliárd forintot fordít.

Kovács Kálmán informatikai és hírközlési miniszter augusztus végén bejelentette, hogy a minisztérium közbeszerzési eljárást indít a Közháló projekt első szakaszának megvalósítására.

A tervek szerint a Közháló programba bevonják majd a jelenlegi Sulinet-végpontokat (iskolákat, kollégiumokat, illetve szakképzési intézményeket), valamint az informatikai tárcához tartozó végpontokat (a pályázati nyerteseket és az eMagyarország-pontokat), és a további bővítések során a kórházakat és a művelődési házakat is. A közháló kiépítésének első szakaszában 2006-ig 2530 településre jut el, ez összesen 7300 végpontot jelent. A közháló egymással összekapcsolt, de önálló egységekből, alhálókból áll össze. *Kovács Kálmán* kiemelte, a közháló nem egy új hálózat kiépítése állami beruházás formájában, hanem a piaci szereplők szolgáltatásainak igénybevétele. Az IHM tervei szerint a Közháló program folyamatosan, több egymásra épülő szakaszban valósul meg. Az első szakaszhoz tartozó feladatok megvalósítására elsőként öt tárgyalásos közbeszerzési pályázat jelent meg a Közbeszerzési Értesítőben az alábbi témakörökben:

- gerinchálózati szolgáltatás biztosítása;
- elérés a gerinchálózati csomópontoktól a kijelölt szolgáltatásig földfelszínen;
- internetszolgáltatás és ügyfélszolgálat, valamint internetlabor-távfelügyelet Sulinet-végpontok számára;
- internet-szolgáltatás és ügyfélszolgálat a nem Sulinet-alhálóhoz tartozók számára;
- hálózatfelügyelet a hálózat távközlési és adatátviteli hátterére vonatkozóan, valamint projektmenedzsment és üzemeltetésfelügyeleti szolgáltatás biztosítása.

Továbbá két nyílt pályázat is kiírásra kerül:

- a VSAT technológiával történő elérés biztosítására,
- az adatkommunikációs berendezések szállítására, telepítésére és karbantartására.

A hét pályázat között lesz olyan is, amelynél a beszerezni kívánt szolgáltatások körének nagyfokú bonyolultsága és a területi felosztás miatt több győztesre számíthatunk. A beszerzések legfontosabb szempontja a megbízható és biztonságos szolgáltatás biztosítása volt. A tárca reményei szerint a hálózati szolgáltatás piaci ár alatti beszerzése a közpénzek hatékony felhasználását eredményezi.

Közös költségvállalás

Az IHM létrehozta az Információs Társadalom Koordinációs Tárcaközi Bizottság (ITKTB) Közháló albizottságát, ami a döntésekben egyeztet a társtárcákkal. Az IHM és az Oktatási Minisztérium (OM) között már létrejött az első szakasz megvalósítására vonatkozó megállapodás. Ennek eredményeképpen a sulinetes végpontok beszerzési és működtetési költségét az IHM és az OM közösen viseli. A közhálóban lépcsőzetes kiépítéssel mintegy ötezer sulinetes végpont, illetve 2300 további végpont kerül bekötésre, így a Sulinet közhálóba kerülésével egységessé válik a civil hálózati rendszer, amelynek a Sulinet mellett további alhálózatai is léteznek. Ez utóbbiak kiválasztásának egyik fő szempontja az volt, hogy a végpontok lehetőleg hátrányos helyzetű területeken valósuljanak meg. Az első szakaszban megvalósuló többi végpont esetében a költségeket az IHM állja. A most indított közbeszerzésekre az IHM és az OM három év alatt várhatóan többmilliárd forintot fordít.

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

„Megabitek a magyar ugaron – hová tartunk Európával?"

Az idén október 28–29-én tartandó Internet Hungary konferencián két napig ismét szakmai viták, beszélgetések és váratlan találkozások helyszíne lesz a tihanyi Balaton-part. Az elmúlt három évben a tihanyi rendezvényen alkalmanként csaknem ezren vettek részt, s ezzel az Internet Hungary a maga nemében a legjelentősebb szakmai fórummá lépett elő Magyarországon. Az elmúlt évek tapasztalatai, illetve az idén eddig befutott több száz jelentkezés alapján a rendezvényen idén is részt vesznek a magyar gazdasági élet jeles képviselői, a marketing-, PR- és kommunikációs cégek vezetői, a médiaszakemberek a sajtó minden területéről, a közvélemény-kutató intézetek, a jogi irodák, az állami hivatalok és a közintézmények vezetői. Ízelítőül néhány a tervezett előadások és fórumok közül: A kockázati tőke bolyongásai; Mitől leszek internetmilliomos? A csatlakozást követően exportőrök vagy importőrök leszünk? Mi lesz az üzleti sláger az elkövetkező időszakban? Hogyan lehet pénzhez jutni, ki és milyen szempontok szerint dönt az állami megrendelésekről, pályázatokról? Az „Internet" szó említése milyen kontextusba került az elmúlt időszakban: érdekli-e a magyar közönséget az Internet, és ha igen, miért nem? A vonalkódos tehén és a csipp-pírszinges disznó: az informatika megjelenése a mezőgazdaságban; Nézetek, és nézetkülönbségek a „netújságírással" kapcsolatban; Lyukak a szabályozásban, törvényi akadályok az információs társadalom útjában; Elszámolóház: hogyan lehet fizettetni a letöltésekért? (Digital Rights Management); Te is, fiam, Blútusz! Verseny vagy kooperáció: Wi-Fi, broadband, mobil Internet.

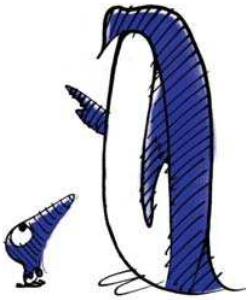
Hol tart az OpenOffice.org?

Talán kicsit megkésve, de el nem feledkezve e kitűnő irodai csomagról, következzék egy rövid tájékoztató az OpenOffice.org 1.1-es magyar változatának állapotáról és a jelenleg is zajló munkálatokról.

Az OpenOffice.org 1.1-es változatának közelgő megjelenése miatt a magyar fordítás elhúzódó lektorálása sürgős feladattá lépett elő. A fordítást a 174 bejegyzett önkéntes fordító rövid idő alatt elvégezte. Az 1.0-s változat fordítása a 2002 februári, nagy sajtóvisszhangot kiváltó három napos maratoni fordítóhétvégén, majd 1.0.3 és az 1.1beta2 változatokra történő frissítés szintén pár nap alatt készült el idén áprilisban és júliusban. A lektorálást azonban ezzel a sok embert megmozgató módszerrel nem lehet elvégezni, hiszen e munkafolyamat célja többek között épp az egységesség megteremtése, ami inkább kis létszámú felkészült csapat feszes tempójú, időben korlátos erőfeszítését igényli. Bár a fordítást követően a lektorálás is elkezdődött, szintén önkéntes alapon, az anyag mennyisége miatt nem készült el. A folyamat elhúzódása és ellenőrizetlensége miatt az elkészült lektorálás minősége sem lett egyenletes. Még a három napos fordítóhétvégén a karakterláncok negyven százalékát lektoráltként jelölték be az önkéntes lektorok, de ezek között is gyakran volt hibás fordítás. Az ezt követő másfél évben többek között *Verók István*, *Somogyi Péter* és jómagam újabb tíz százalékot tettünk hozzá ehhez, de mivel a fordítórendszerben lektoráltként megjelölt karakterláncokban sem lehetett százszázalékosan megbízni, az FSF.hu Alapítvány a teljes anyag (újra)lektorálását tűzte ki célul. Megbízható önkéntes vállalkozó hiányában erre a feladatra profi fordítócéget kellett felkérni.
Talán sok olvasó tudja, hogy volt (illetve van) egy bizonyos pénzkeret az OpenOffice.org honosítására, amelyet először a Miniszterelnöki Hivatal pályáztatott meg, és az összeget a Linux-felhasználók Magyarországi Egyesülete kapta meg. Ezt követően az LME írt ki egy pályázatot, de a pályázaton nyertes cég nem volt képes a munkát elvégezni. Az FSF.hu által szervezett fordítóhétvégén végül is elkészült a fordítás, és született egy megállapodás az FSF.hu és az LME tisztségviselői között, hogy a maradék pénzt az LME átutalja az FSF.hu-nak. A felek kikötötték, hogy a pénzt az OpenOffice.org honosítására kell költeni. Az átutalásra idén tavasszal került sor. Ilyen módon az FSF.hu Alapítvány 1,6 millió forinthoz jutott hozzá. Ebből fedezték a hamburgi OpenOffice.org-konferencia négyfős magyar küldöttségének költségeit 189 289 forint értékben. Az FSF.hu a maradék 1,4 millió forint egy részéből hivatásos programhonosítókkal és lektorokkal „tettette rendbe" a programot. Ez az előfeltétele az új súgó jó minőségű lefordításának, de a program hatékony használatának is, ugyanis számos olyan fordítási hiba fordult elő benne, ami bizonyos szolgáltatások használatát igencsak megnehezítette.
A lektorálási munkálatokat a Gamax Kft. 2003. augusztus 11-én kezdte el. A befejezés szerződésben rögzített határideje 2003. szeptember 23. volt, de egy héttel előbb lett kész. Összesen 72 000 szót, 6 Ft/szó + áfa áron lektoráltak (540 ezer Ft). A leadott anyag az OpenOffice.org 1.1 (645) szövegeinek adatbázisa. Három lektor és két tesztelő dolgozott rajta; hárman közülük (timar2@, tim@ és ffodor@) hozzáférést kaptak az FSF.hu Alapítvány által üzemeltetett OpenOffice.org fordítófelülethez, az általuk javított karakterláncok száma a „Toplista" oldalon követhető (➲ http://office.fsf.hu/trans/index.php?showtoplist=1).
A lektorálás kezdetén elkészült egy 1139 bejegyzésből álló szószedet (➲ http://office.fsf.hu/hun_gloss.html), amely a program legfontosabb kifejezéseinek magyar fordítását adja meg. A szószedetet az FSF.hu Alapítvány szakemberei elfogadták. Bizonyos programrészek fordítása eleve elfogadható minőségű volt, de sok részt gyakorlatilag újra kellett fordítani. A program felhasználói felületét alkotó körülbelül 21 ezer karakterlánc nagyjából egyharmadához kellett így vagy úgy hozzányúlni.
A lektorálás alatt rendszeresen kikerült az ➲ office.fsf.hu honlapra egy csomag RES-fájl. A jelenlegi állapot a ➲ http://office.fsf.hu/work/TAGGED-res-1.1rc3-gamax-20030915.zip címen tekinthető meg, ami pillanatkép a lektorálás állásáról. Semmi másra nem alkalmas, csak a magyar fordítás kipróbálására. A fájlokat az OpenOffice.org 1.1 645-ös (RC) összeépítésének (buildjének) *Program/Resource* könyvtárába kell másolni, felülírva az ott található angol nyelvű fájlokat. Mivel a gamaxos összeépítés a Ximian összeépítésén alapul, nem ximianos összeépítés alól nézve az átlátszó hátterű 16 millió színű ikonok helyett néhány helyen csak fekete kockákat láthatunk. Ezzel sajnos meg kell békélni, a Gamax Kft-nek nem volt feladata egy teljes értékű összeépítés elkészítése. Az FSF.hu Alapítvány úgy döntött, hogy a végleges magyar változat – amelyet az Alapítvány aktivistái készítenek majd el – a legutolsó 1.1-esen alapuljon; nem foglalkoznak a fejlesztői és a próbaváltozatokkal. A végleges OpenOffice.org 1.1 a cikk írásának pillanatában még nem jelent meg.
Bár a lektorálás céges keretek közt zajlott, a projekt reményhetőleg nem veszítette el teljesen közösségi jellegét. Az FSF.hu Alapítvány és a Gamax Kft. számít a felhasználók visszajelzéseire. Várják a hibajelentéseket és a javaslatokat akár a Bugzillába (➲ http://kac.poliod.hu/bugzilla), akár elektronikus levélben a timar@gamax.hu címre. Elsősorban olyan hibákra kíváncsiak, amelyeket a felhasználók a program használata közben találnak, különös tekintettel félrefordításokra vagy az oda nem illő fordításokra (természetesen a 4 betűs azonosító és a kérdéses programrész elérhetőségének megadásával).

Tímár András (timar@fsf.hu)
Az FSF.hu Alapítvány kurátora és a Gamax Kft. programhonosítási csoportjának munkatársa. Szabadidejét legszívesebben családjával tölti.

Várják a hibajelentéseket és a javaslatokat akár a Bugzillába ➲ http://kac.poliod.hu/bugzilla, akár elektronikus levélben a timar@gamax.hu címre.



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

Red Hat hálózati kártya nélkül?

Lehetséges-e hálózati kártya nélkül futtatni a Red Hat 8.0-s Linux-változatot?
Prasanna, softpars@rediffmail.com

Természetesen igen, és ehhez semmit sem kell tenned. Minden normális Linux-rendszer magától beállítja a lo (visszacsatolási hálózat) eszközt. Még enélkül is képes futni a Linux (beágyazott rendszerekben néha így fut). A lényeg, hogy semmi különöset sem kell tenned. Egyszerűen telepítsd a Linuxot, és ne adj meg hálózati beállításokat.
Jim Dennis, jimd@starshine.org

Meleg van odabenn

Asus alaplapot használok. Microsoft Windows 98 alatt az asusprobe program szerint a processzor hőmérséklete 47 °C, az alaplapé 1 °C. Az értékek kismértékben változnak, de eléggé kiegyensúlyozottak. Hogyan olvashatom ki ezeket a hőmérsékletértékeket Linux alatt?
Michael Mather, mmather@eol.ca

A legtöbb kereskedelemben kapható alaplap az LM78 lapkakészlet köré épülő megoldás segítségével adja át ezt az adatot az i2c kéteres sínen keresztül. Az SMBus az i2c egyik megvalósítása. Linux alatt az `lm-sensors` csomag tartalmazza az ezeknek az adatoknak az elérését biztosító illesztőprogramokat és segédprogramokat. További tudnivalók a csomagról a ➲ http://secure.netroedge.com/~lm78 címen találhatók. Az `i2c` és az `lm-sensors` illesztőprogramjai részei a rendszermag forrásának, és a nagy Linux-terjesztésekbe (pl.: Red Hat, Debian és SuSE) belefordítják. A GYK tanulmányozása során kiderül, hogy különböző alaplapok különböző számokat adnak vissza a hőmérséklet és a feszültség értékeiként. A visszaadott számok helyes értelmezése a */etc/sensors.conf* állományban állítható be. A Debian egy körülbelül húszoldalas *sensors.conf* mintaállományt telepít. A GYK külön említi az Asus P2B alaplapot, mint olyat, ami furcsa hőmérséklet értékeket ad vissza. Ha ilyen alaplapod van, olvasd el a GYK-t a fenti címen. Mellesleg rengeteg program használja az `lm-sensors` által nyújtott felületet. Vannak démonok, amelyek összegyűjtik az adatok idősorait – ezeket később elemezhetjük, illetve kirajzolhatjuk, és vannak grafikus kijelzők, amelyek a KDE, Gnome, illetve a Window Maker paneljébe épülnek be. A legegyszerűbb a `sensors` program, ami beolvassa a */etc/sensors.conf* állományt, és az illesztőprogramtól kapott nyers adatokat ennek megfelelően alakítja át.
Jim Dennis, jimd@starshine.org

Összeférhetetlen webhely?

Nemrég a Konqueror 2.2.2 programmal ellátogattam egy hitelkártyacég weblapjára, hogy ellenőrizzem az egyenlegemet. A webhely közölte, hogy frissítsek legalább Netscape 4.0-ra vagy Microsoft Internet Explorer 4.0-ra, hogy használni tudjam a 128 bites titkosítást, amit ők használnak. Úgy tudom, hogy a Konqueror 2.2.2 sokkal újabb, mint az említett böngésző 4.0-s változata. Megnéztem a böngésző beállításait, és meggyőződtem róla, hogy az SSL2 és SSL3 a 168 bites titkosításig bezárólag engedélyezett. Majdnem minden titkosítási szabvány engedélyezett volt, kivéve a FZA-FZA-CBC-SHA, FZA-NULL-SHA, NULL-MD5 és NULL-SHA – ezek „0 of 0 bits" beállítással szerepeltek. Végül beállítottam a Konquerorban, hogy Netscape-nek vagy MSIE-nek hazudja magát, erre a hibaüzenetek varázsütésre eltűntek. Az űrlapok – például a bejelentkezési képernyő – kitöltése után viszont nem történik semmi, az űrlap frissíti magát, de az adatokat nem dolgozza fel. A JavaScript, a Java és a sütik engedélyezettek. Ennek ellenére sehogy sem működik. Én csinálok valamit rosszul, vagy a webhely nem szabványos? Amikor a beállítások között megnéztem a tanúsítványokat, ott volt ennek a webhelynek a tanúsítványa is. A tanúsítványt ellenőriztem, de nem derül ki belőle, hogy milyen titkosítást használnak, és nem ad más használható adatot sem.
John Handis, mrintensity@worldnet.att.net

Túl sok webfejlesztő ír kódot egy adott megvalósításra (célfelületre) ahelyett, hogy szabványos protokollokat és API-kat használna. A JavaScript tovább súlyosbítja a helyzetet. Csüggesztően nehéz olyan kicsit is összetett JavaScript-kódot írni, ami minden böngészővel helyesen működik. Minden webfejlesztőt arra buzdítok, hogy a webhely alapfeladatát egyszerűen valósítsák meg. Ezután JavaScriptben mindenféle csicsát hozzá lehet adni, de az alapfeladatot ezek nélkül is végre kellene tudnia hajtani. Javaslom, hogy tegyél panaszt a banknál, és próbáld ki Mozilla 1.x vagy Netscape 6 böngészőkkel is.
Jim Dennis, jimd@starshine.org

Minden rosszban van valami jó. Talán még nem hallottál róla, de az Apple alapértelmezett webböngészője, a Safari a KDE projekt KHTML megjelenítőmotorján alapul. Ez természetesen ugyanaz a motor, amelyet a Konqueror is használ. Mivel a Microsoft abbahagyta az Internet Explorer fejlesztését a Macintosh-felületen, a csak Windowsra és Macintoshra fejlesztő webmesterek kódja a te böngésződdel is működni fog.
Ben Ford, ben@kalifornia.com

Sok kép gyors letöltése

Tegyük fel, hogy van egy ➲ http://www.foo.org/technical/pics nevű webhely. Hogyan tölthetem le csak a képeket – tegyük fel, hogy a kiterjesztésük *.jpg* – a `wget` segítségével?
Kunthar, kunthar@gmx.net

Íme egy példa:

```
wget -r -l1 --no-parent -A "*.jpg"
http://www.server.com/dir/
```

Ez minden JPG-fájlt letölt a ➲ http://www.server.com *dir* könyvtárából. A további lehetőségeket a `wget` súgóoldalán (man wget) olvashatod el.
Felipe Barousse Boué, fbarousse@piensa.com

Új termékek

AlterPath ACS1

Az AlterPath ACS1 olyan kisméretű egykapus konzolkiszolgáló, amellyel soros eszközök csatlakoztathatók a TCP/IP hálózatra. A jellegzetes felhasználási területek között megtalálható a fiókiroda-kezelés, az értékesítés gépesítése és a régi soros eszközök IP-hálózatba kötése. A két PCMCIA bővítőhely segítségével az

eszköz tudása kiterjeszthető. Sokféle illesztőkártya csatlakoztatható, például ethernet, modem (V.90, GSM, CDMA és ISDN), valamint vezeték nélküli LAN. Az ACS1 két PowerPC processzort használ a 10/100-as ethernetcsatoló és az RS-232/RS-485 soros csatoló közötti adatátvitelhez, így valósul meg a soros eszköz és a hálózat összekötetése. Az ACS1 az SSHv2 segítségével támogatja az adatkapcsolatok titkosítását.

Adatok: Cyclades Corporation, 41829 Albrae Street, Fremont, California 94538,
➲ http://www.cyclades.com

ATCA-710 SBC

A Force Computers bejelentette új egykártyás számítógépét, ami az AdvancedTCA nyílt szabványsorozaton alapul. Az 1,8 GHz-es Pentium 4 processzort tartalmazó ATCA-710 támogatja a rendkívül nagy rendelkezésre állást igénylő Linux-alkalmazásokat, valamint a 2,6 G/3 G vezeték nélküli és a széles sávú vezetékes hálózatokat. Választható kiegészítő a négy PMC-bővítőhelyet kínáló bővítőkártya, ami PCI-X-es I/O-bővítőkártyákat képes fogadni, valamint egy 12 kapus ethernetkapcsoló. A lapkakészlet 1,6 MB/s memória-sávszélességet, 266 MHz-es ECC (hibajavító) DDR SDRAM-ot, legfeljebb 8 MB felhasználói és 4 MB rendszerindító flashmemóriát támogat.

Adatok: Force Computers, 4211 Starboard Drive, Fremont, California 94538,
➲ http://www.forcecomputers.com

SciTech MGL 5.0

A SciTech MGL 5.0 az X Window System helyett választható alacsony szintű grafikus programkönyvtár Linux, Unix, QNX, OS/2 operációs rendszer és beágyazott rendszerek számára. A SciTech MGL C és C++ nyelven írt grafikus kódot tartalmaz, ami többféle operációs rendszeren működő alkalmazások készítésére használható. A SciTech MGL elvégzi a felhasználói programokat, játékokat vagy valós idejű grafikus alkalmazásokat felépítő síkbeli és térbeli alapelemek gyors, alacsony szintű leképezését. A SciTech MGL segítségével létrehozott alkalmazások az OpenGL API-t használják ablakban vagy teljes képernyős módban, és megjelenhetnek VGA, VESA VBE vagy DirectDraw felületeken. A programkönyvtárra az LGPL és kereskedelmi felhasználási engedély vonatkozik.

Adatok: SciTech Software, Inc., 180 East 4th Street, Suite 300, Chico, California 95928,
➲ http://www.scitechsoft.com

Big Medium 1.1

A Big Medium 1.1 egy webalapú tartalomkezelő program, amely lehetővé teszi, hogy a HTML nyelvet nem ismerő felhasználók is írhassák, szerkeszthessék és közzétehessék a webes tartalmakat. A Big Mediumot a Unixon és változatain futó webkiszolgálókhoz tervezték. Gyorsan felállítható egy új webhely az előre beállított arculat, elrendezés és tartalom alapján. Jelenleg a hírekhez és a termékek bemutatásához tartalmaz sablonokat, de nemsokára máshoz is készülnek sablonok. A Big Medium képes megkettőzni és menteni az általa létrehozott webhelyeket; a lektor üzemmódban böngészhetjük a webhelyeket, tetszőleges tartalmat szerkeszthetünk át vagy törölhetünk; egy meglévő webhelyre alkalmazhatjuk a sablonok egyikét. A Big Medium Perlben íródott és szabványos HTML/XHMTL-kódot használ. Az ingyenes próbaváltozat a ➲ http://demo.globalmoxie.com címről tölthető le.

Adatok: Global Moxie,
➲ http://www.globalmoxie.com

In-Reach LX-4048S

Az MRV Communications bejelentette az In-Reach LX-4048S-t, amely egy, a távoli rendszerfelügyeletet segítő biztonságos konzolkiszolgáló. Az LX-4048S operációs rendszere az Iros (In-Reach Operating System). Ez egy Linux alapú operációs rendszer, amelyet a teljesítményre, biztonságra és megbízhatóságra hangoltak. Az In-Reach 48 kapuval bír, nagy adatközpontok és fürtözött számítóközpontok felügyeletére és vezérlésére tervezték. Az In-Reach biztonságos távoli elérést tesz lehetővé. A soros kapcsolódási lehetőség, az energiaellátás és riasztások kezelését egy készülékkel oldhatjuk meg. A belső modem minden kapuhoz használható. Mindegyik változat 32 bites RISC processzort használ. A biztonság többek közt a kapunként megadható jelszó, RADIUS, PPP-visszahívás segítségével és még sok más módon szavatolható.

Adatok: MRV Communications, Inc., 20415 Nordhoff Street, Chatsworth, California 91311,
➲ http://www.mrv.com

ProStore SATA

A ProStore SATA (soros ATA) tárolórendszer több terabájt tárhelyet biztosít a sok adattal dolgozó alkalmazások – például fürtözés, digitális tartalom előállítása és tudományos megjelenítés – számára. A ProStore 36 soros ATA-meghajtót tartalmaz egy 4U magas házban, és a jelenlegi SATA-meghajtók nagyságát figyelembe véve legfeljebb 9 TB tárhely építhető ki. A ProStore-ban 200 GB-os SATA-meghajtók találhatók, a lapkakészlet Intel E7501, tartalmaz két Xeon 3,06 GHz-es processzort, és legfeljebb négy 3ware Escalade 8500 soros ATA RAID-vezérlőt. A merevlemezeket és tápegységeket működés közben is cserélni lehet, van benne két 10/100/1000 ethernetcsatoló, a különféle RAID-szintek támogatása miatt a rendszer hibatűrő és megbízható.

Adatok: ProMicro, 13880 Stowe Drive, Poway, California 92064,
➲ http://www.promicro.com

Linux Journal 2003, 113. szám

A Nyíltforráskód-fejlesztői Labor

Mire jó az OSDL azon kívül, hogy alkalmazza Linus Torvaldsot?

Amikor egy műszaki megoldás címlaptörténetté válik egy vezető üzleti magazinban, biztosak lehetünk benne, hogy végre bekerült a köztudatba. Az utóbbi időben a Linuxra irányuló sajtófigyelem alapján könnyű lenne feltételezni, hogy már meg is nyertük a közismertségért vívott csatát, és mostantól végre könnyebben mennek majd a dolgok. De mi, akik az üzleti számítástechnika frontvonalában dolgozunk, tudjuk, hogy az igazi kulimunka csak most kezdődik.
A pingvin egy ideje már a lehetőségek és a programfejlesztés gyökeres megújulásának a jelképe, de most jött el az ideje, hogy valóra is váltsa a benne rejlő ígéreteket.
Az a tény, hogy az üzleti életben egyre szélesebb körben kezdik komolyan venni a Linuxot, kedvező fejlemény: többet foglalkoznak a Linuxszal és több fejlesztői erőforrást fordítanak a cégeknél a linuxos programokra. Ugyanakkor ezzel a fokozódó figyelemmel együtt járhat egyfajta bizonytalanság is – főleg a nyílt forráskód közösségével kapcsolatban kevés tapasztalattal rendelkező emberek és cégek részéről. És ha van valami, amit a vállalatok informatikai osztályán nem szeretnek, az a bizonytalanság. Ahogyan a Linux elmozdul a hálózathatároktól az adatközpontok felé, a linuxos programok egyre nagyobb terhelésnek lesznek kitéve, és a hibák reflektorfénybe kerülnek.
Annak ellenére, hogy webkiszolgálóként nagyon elterjedt, valamint pezsgő és elkötelezett fejlesztői közösségének köszönhetően kiforrott, a Linuxnak valódi vállalati felületként még bizonyítania kell. Az informatikai részlegeknél meg kell győződniük róla, hogy a Linux biztonság, méretezhetőség és rendelkezésre állás tekintetében a jogdíjas rendszerekkel egyaránt összemérhető. A független és a cégeknél dolgozó programfejlesztőknek olyan ellenőrző eszközökre van szükségük, amelyek egy vállalati adatközpontot megjelenítve lehetővé teszik a kód ellenőrzését és a megfelelő javítások elvégzését. Itt lép színre a Nyíltforráskód-fejlesztői Labor (Open Source Development Laboratory, OSDL).
Az OSDL egy nonprofit vállalat, amit 2000 augusztusában műszaki cégek hoztak létre azzal a céllal, hogy felgyorsítsa a Linux terjedését a vállalati számítástechnika terén. A labor olyan hely, ahol elvégezhető a rendszermag és a köztes kód terheléses próbája és megerősítése a linuxos vállalati alkalmazások támogatása céljából. Az alkalmazásfejlesztés támogatása mellett tevékenyen részt veszünk a Linux-rendszermag és a köztes programrétegek fejlesztésében is. Az OSDL nemrég új eszközkezelő modullal járult hozzá a Linux 2.5-ös maghoz, és komoly munkát vitt véghez a megbízhatóság javításában is.
Az OSDL azért jött létre, hogy a nyílt kód fejlesztői alkalmazásaik ellenőrzéséhez egy adatközpont-jellegű környezethez férhessenek hozzá, és műszaki, valamint erkölcsi támogatást kapjanak. Bár az OSDL tevékenységi köre kiszélesedett az elmúlt évben, az eredeti küldetés központi eleme változatlan maradt: erőforrásokat és útmutatást biztosítani a nyílt kódú alkalmazások fejlesztőinek ahhoz, hogy adatközpont- és távközlési szintű megoldásokat építsenek a Linuxba és annak nyílt kódú programkészletébe.
Az OSDL három különböző programot nyújt a fejlesztőknek,

elősegítendő a Linux vállalati elterjedésének és elfogadottságának felgyorsítását:

1. Teljes értékű adatközpont-környezet linuxos fejlesztéshez és kipróbáláshoz, amihez a világ minden tájáról hozzáférhetnek a szakemberek.
2. Vállalati szintű igényeket kielégítő fejlesztőeszközök és teljesítménymérő készletek a cégek, szerződéses terjesztőpartnereik és más Linux-fejlesztők számára.
3. Olyan globális kezdeményezések befogadása és összehangolása, amelyek követelményeket határoznak meg és megerősítik a Linuxot a távközlési és az adatközpont-környezetekben támasztott megbízhatósági, rendelkezésre állási és teljesítménykövetelmények elérése céljából.

Létrehozása óta az OSDL több mint kétszáz linuxos fejlesztést támogatott: az Apache-tól a virtuális memóriakezelés javításáig. A labor tesztkészletét x86 és Itanium rendszerek alkotják, 32 utas kiépítésig. Az erőforrások új és már meglévő linuxos fejlesztések számára is elérhetők.
A fejlesztőeszközök közé tartozik egy önműködő, méretezhető próbafelület (Scalable Test Platform – STP) a Linux számára, amivel ismételhető próbák során ellenőrizhető, hogyan teljesítenek a foltok és fejlesztések vállalati informatikai környezetben. Rendelkezésre áll egy foltételciklus-követő rendszer is, amivel az STP kipróbálás előtt ellenőrizhető, hogy a folt a Linux-rendszermaggal lefordítható-e. Meg akarjuk mutatni a vállalati felhasználóknak, hogy milyen módon segítheti a nyílt forráskód az üzleti tevékenységüket, és azt is, hogyan válhatnak a közösség cselekvő tagjaivá. Mint látható, kijut nekünk a munkából. De jut másnak is. Mindenkit meghívok a ➲ http://www.osdl.org honlapra – nézze meg, miben mesterkedünk. Talán hozzá is tehet valamit.

Linux Journal 2003, 113. szám

Stuart Cohen
A Nyílt Forráskód Fejlesztői Labor ügyvezetője.

A Linux szerepe a Wi-Fi New York-beli kialakulásában

Közösségek, induló vállalkozások, sőt még a telefontársaságok is a Linuxot használják, hogy New York városát egyetlen hatalmas és boldog vezeték nélküli hálózati hozzáférési ponttá tegyék. Mi a helyzet a te városodban?

A nyilvános vezeték nélküli hálózatok a betyár- (hacker) közösség nyúlványai. Számukra ez egy olyan módszer, amellyel a széles sávú internetet el lehet vinni a nyilvános helyekre. Az utcán és a parkokban sétáló felhasználók számára pedig a közösségi élet egy sokkal civilizáltabb módja. A nyilvános Wi-Fi a szolgáltatóktól való függetlenséget hozza magával, és az internetet ajándékká teszi, áldássá a polgárok számára, mely áldás a parkok, a gyalog- és sugárutak, valamint a könyvtárak közelében érhető el.
2003 májusában az FCC folytatta deregulációs tevékenységét az úgynevezett „nyilvános" frekvenciák tulajdonlása terén, s a szervezet nagyon sokat tett a még megmaradt „ingyenes vezeték nélküli műsorszórás megmentése" érdekében. Az internetnek azonban nincs szüksége sem deregulációra, sem szabályozásra ahhoz, hogy ingyenes legyen az éterben. Mindaz, amire szüksége van: nagylelkű műszakiak, polgárok és civil szervezetek. Ez az, amire ma New Yorkban számíthatunk. A közreműködők munkája pedig figyelemreméltó.

A behálózott New York

A nyilvános Wi-Fi-kísérleteket többnyire a városháza kezdeményezi. Ez történt a kaliforniai Long Beachben, ahol óriási nyilvános „vezeték nélküli zónát" üzemeltetnek: egyet a belvárosban, egy másikat pedig a reptéren. Más kezdeményezéseket a hozzáértő önkéntes műszaki érdeklődésű emberek irányítanak, például Austinban, Londonban, Perthben, Seattle-ben és San Franciscóban. A cégek szintén hozzáteszik a magukét. Az északkarolinai Asheville-ben a Natural Communications nevű vállalat nyilvános vezeték nélküli elérési pontot kínál, amit BeamPostnak neveznek. New Yorkban azonban másképp állnak a dolgok: a fent említett módszerek mindegyikét alkalmazzák.
Bár New York csak a 27. helyen áll az Intel által készített „legbehálózottabb" városok listáján (az oregoni Portland áll az élen), valószínűleg ez a legjobb példája a betyárok, a vállalkozások, a kormányzat és a nonprofit szervezetek hatékony együttműködésének a nyilvános és ingyenes Wi-Fi iránti igény kielégítésében. Ez az együttműködés indította útjára a NYCwireless projektet, amelynek tagjai „érdeklődő elmék szabad gyülekezete"-ként jellemzik magukat. A NYCwirelessnek két küldetése van: ingyenes és nyilvános vezeték nélküli internet-hozzáférést, valamint fórumot biztosítani a vezeték nélküli technológia fejlődése számára.
A NYCwireless alapítói, ***Anthony Townsend*** és ***Terry Schmidt*** résztulajdonosa az Emenitynek, annak a cégnek, amelyik az új NYCwireless-hátteret építi ki New Yorkban. A NYCwireless és az Emenity a nyilvános és a magánszféra együttélésének terméke. Ugyanez vonatkozik a vevőikre is, akik között megtalálhatók a nyilvánosan alapított szomszédközösségek. Ezeket többek között azzal a céllal hoztak létre, hogy olyan hátteret alakítsanak ki, mint például a parkokban jelenlévő nyilvános Wi-Fi.
Májusban New York városának közgyűlése kiadott egy jelentést, amely a város elaprózódó, széles sávú eszközök beszerzési eljárásának átdolgozását javasolja és egy új üvegszálas, illetve vezeték nélküli városi hálózat (MAN), valamint egy nyilvános Wi-Fi hálózat kialakítását kezdeményezi. Az utóbbinak lehetne az egyik megvalósítása a még csírájában létező Prospect Park Wi-Fi hálózat, amelynek a kialakítása nagyjából 192 ezer dollár lenne, a fenntartására viszont keveset kellene fordítani. A jelentés egy Anthony Townsendnek való köszönetnyilvánítással kezdődik, aki a New Yorki Egyetem (NYU) városkutató központjának kutatója, ahol rengeteg bölcs és előremutató tanulmány-t írt az internetnek a városi környezetben történő növekedéséről. Terry Schmidt feladata Anthony víziójának a megvalósítása. Terry az Emenity műszaki igazgatója és a Pebble Linux mögött álló szakember. Ez a Linux-változat egy lecsupaszított Debian, és a NYCwireless hozzáférési pontjaiban használják.

Jelhalászatra indulva

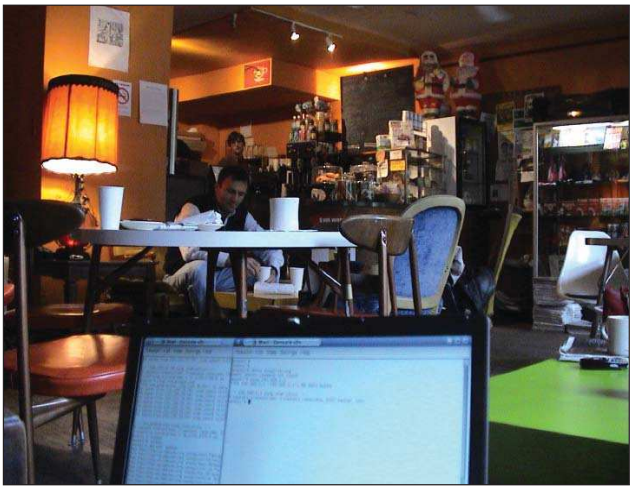
A Wi-Fi hatóköre nagyon kicsi. A jogdíj nélkül használható mikrohullámú frekvencia szűk kis szeletében működik, amelyet a 2,412 és a 2,484 GHz közti tartományban 14 csatornára osztottak. Az Államokban csak az 1–11-es csatornát használják, Európában az 1–13-asig, kivétel Franciaország, ahol a 10–13-as csatornákat használják, Japán pedig az 1–14-es csatornákat alkalmazza. A hozzáférési pontok (más néven AP, WAP vagy alapállomások) alapértelmezett átviteli teljesítménye 30 mW, ez körülbelül az egytizede a mobiltelefonok teljesítményének. Magasabb frekvenciát alkalmaz, ahol az energia a távolság függvényében sokkal nagyobb mértékben gyengül a levegőben, és sok épületen nem tud keresztülhatolni, ilyen anyagok például a thermoablakok és a vizes falevelek, amelyek elnyelik a mikrohullámot.
A Wi-Fi hatótávolsága kisebb, mint a sokkal erősebb jelet szolgáltató átlagos vezeték nélküli telefonoké. Egy ilyen ügyes, kis hatótávolságú szolgáltatásnál természetesen oda kell a legerősebb jelet összpontosítani, ahol nagy a népsűrűség, és könnyen „hozzáférhető" a lakosság. Ilyen hely New York, ahol az emberek egymás hegyén-hátán élnek és dolgoznak.
Hat jelkutató utazást hajtottam végre, ez kilenc taxiutat foglalt magában, a többségükre Manhattanben került sor. Az utolsó utam az autópályán vitt keresztül a LaGuardia reptérre a jelmentes Queens részein keresztül. Mindegyik utam során rögzítettem az észlelhető jelek alapadatait, az ESSID-et (Extended Service Set Identifier – kiterjesztett szolgáltatásazonosító).
A kilenc út során összesen 1548 nyitott hozzáférési pontba jelentkeztem be.
Manhattan utcáin szinte mindig hatótávolságon belül voltam, és meggyőződésem, hogy a fent említett szám még nagyobb lett volna, ha minden taxin lett volna egy külső antenna – a hátsó ülésen lévő laptopom beépített antennája helyett. Bár nagyon sok kereskedelmi hozzáférési pont létezik, úgy tűnik,

hogy a túlnyomó többségük magánemberekhez tartozik. A legtöbb ESSID „Linksys” típusú, ami nagyon népszerű és olcsó hozzáférési pont.
Hihetetlen mértékű az egyének hajlandósága a sávszélesség megosztására. Bár a teljesen nyitott AP-k száma kevesebb, mint azt a WEP-ek száma előre jelezte volna. Meglehetősen sok jelszóval védett volt, de még így is sok használható jelforrás maradt. Többször is sikerült elkapni egy jelet és levelet küldeni, miközben egy piros lámpánál álltunk a taxival.
A jelhalászat eszményi módja egy Kismet programot futtató linuxos vagy BSD-s laptop. Az alkalmazás egy vezeték nélküli hálózati szaglászó program. Olyan sok szolgáltatással bír, hogy még egy olyan ügyes kis dologra is képes, mint GPS segítségével pontosan meghatározni a jelforrás helyét.

Utastársaim

Pontosan az utam előtt említettem meg egy SuitWatch-hírlevélben, hogy New Yorkba jövök tanulmányozni a Wi-Fi helyzetét, és kértem egy kis helyi segítséget. Az első válasz ***Kurt Starsinic***-től érkezett, aki gyorsan személyes Wi-Fi-docensemmé vált alsó-manhattani jelkutató utazásaim és sétáim során. Kurtöt az A. Avenue-i Alt.Coffee-ban fogtam be a szemben lévő Tompkins Square Parkból. Az Alt.Coffee egy kissé lerobbant kávéház, egyben öreg számítógépek ereklyeőrző helye. Kaypros, ARCnet jelelosztók (hub), korai évjáratú PC-k és más antik tárgyak találhatók szétszórva az asztalokon és egymásra halmozva a sarkokban. Érdemes megnézni.
Ahogy az lenni szokott, ottlétem alatt sem az Alt.Coffee, sem pedig a NYCwireless AP-ja nem működött; amikor azonban a Tompkins Square Parkban sétáltunk, találtunk egy otthoni, nyitott és használható kapcsolattal rendelkező csomópontot – ezt természetesen használtuk is.
Következő megállónk a City Hall Park volt, ahol a NYCwireless jelei tisztán és erősen foghatók. Itt alkalmam nyílt igazán kipróbálni. Miközben mindketten régi barátom, ***Stephen Lewis*** felbukkanására vártunk, Kurt rövid eligazítást adott technológiai kérdésekről.
Steve amerikai és holland kettős állampolgár, egy európai telekommunikációs vállalat veteránja, és nagyon kíváncsi volt, hogy mi történik a Wi-Fi-vel a szülővárosában. Miközben Steve régi törzshelyeinél sétáltunk a Lower East Side-on, egyszerűen lenyűgözött minket a Wi-Fi-sűrűség – mind a nyilvános, mind pedig a magánforrások: a Verizon nyilvános telefonfülkéi, a McDonald's éttermei, a Starbucks kávézói és a magánlakások.
Annak lehetősége, hogy egy város kültéri környezetéből szinte bárhonnan rá tudjunk kapcsolódni a webre, különleges élmény megragadó volt Steve számára. Steve kétszeres Fulbright ösztöndíjas, és minden információra ki van éhezve. Ennek eredményeképpen becsvágyó terveket kezdett szövögetni, hogy a New York Wi-Fi környezetének példáját (beleértve a Linux-módszereket is) Bulgáriában is bevezesse, ahol az utóbbi évtized nagy részét töltötte.
A New York-i Wi-Fi-mozgalom egyik legérdekesebb alakja ***Drazen Pantic***, aki korábban a Belgrádi Egyetem matematika-professzora volt. Drazen működtette a B92 internetszolgáltatását, egy olyan rádióét, ami szálka volt a Milosevics-rezsim szemében. Miután a rádióállomás adója rejtélyesen elnémult, Drazen biztosította az adó hír- és adatszolgáltatását, ami ezután az adó honlapján jelent meg. Mindemellett rádiófolyam-szolgáltatást is adott, ezt az Egyesült Királyságban, Hollandiában, az Egyesült Államokban és nem utolsósorban Jugoszláviában újra sugározták. Az adó analóg jeleit Hollandiában fogták be, majd egy műholdon keresztül sugározták újra őket. Ennek eredményeképpen a B92 egyhamar az elsődleges hírforrássá vált Jugoszláviáról és Jugoszláviából, valamint az ott zajló összecsapásokról. Amikor Drazen élettörténetét hallgattam, egyértelművé vált számomra, hogy számos forradalomnak volt egyidőben a hőse.

1. kép Kurt Starsinic, a vezeték nélküli helyek kalauza



2. kép Az Alt.coffee egy linuxos laptop mögül nézve

3. kép
A City Hall Park üdvözlőoldala bemutatja a közeli nevezetességeket

Pebble Linux: Debian a Wi-Fi hozzáférési pontokhoz

Ha fel akarsz állítani egy nyilvános hozzáférési pontot (AP), akkor szükséged lesz valamire, ami jól működik és jól irányítható, továbbá kisméretű és megbízható rendszer. Valamire, ami természetesen a Linux egy

Egy költségkímélő, egykártyás (Single-Board) számítógép a Soekris Engineeringtől

változatán fut. A Pebble Linux egy apró Debian alapú Linux, amely a „betölt és indít" alapját adja. Ez egy (viszonylag) teljes terjesztés, könnyen testreszabható, az AP-nek nincsenek mozgatható részei. Az NYCwireless *Terry Schmidt* készítette, és egy csapat betyár (hacker) és felhasználó tartja karban. A Pebble egy Debian, amit olyan méretre csupaszítottak le, hogy kényelmesen elférjen a 128 MB-os flashkártyára. Mivel ez Debian, viszonylag könnyű hozzáadni és leszedni róla csomagokat. Íme annak leírása, hogy Terry Schmidt hogyan oldotta meg: „Leszedtem a dokumentációt, az összes Perl-cuccot, a binárisok (binaries) nagy részét, az összes csomagot, amiről úgy gondoltam, hogy nem létszükséglet. Lementem 44 MB-ra. Egy igazi terjesztés tudását akartam, például a Debianét, olyan méretben, ami ráfér egy CompactFlashre egy olyasvalamiben, mint egy Soekris doboz. Megoldottam az `apt-get` install `apache`-t és puff, ott az Apache. Így megvan a csomagkezelő az összes szolgáltatásával, amit csak elvárasz tőle."
Terry README-fájljában még ez olvasható (➲ http://www.wireless.net/pebble/pebble.README): a legnagyobb előny, hogy írásvédettként (read-only) megy fel minden. Nem kell sokat aggódnod a CompactFlash megtelése miatt, sem a megfelelő kikapcsolás miatt. Dugd be és húzd ki, ahányszor csak akarod.
Két olyan csomag van egy alapvető Pebble-lenyomatban, amelyek nincsenek Debian-csomagként telepítve:

- HostAP: a Prism alapú 802.11 kártyák meghajtója, amely – szemben egy ügyféllel – a legkedvezőbb módon támogatja a futó AP-t. Ez megszerezhető Debian-csomagként, de a Pebble a CVS legutóbbi változatát használja, mivel az jobban társul a 802.11g ügyfélhez.
- NoCatAuth: a további részleteket lásd az alábbiakban!

Három választható csomag létezik:

- „Pebble-háló"-támogatás, amely lehetővé teszi, hogy több Pebble-gép egy átlátszó hálót alakítson ki. Ez azt jelenti, hogy egy felhasználó anélkül tud barangolni, hogy megváltoztatná az IP-címét vagy elveszítené a hálózati hozzáférést. Az AP hálóképesség (ami hihetetlenül király) a legérdekesebb adalék azok számára, akik nyilvános, vezeték nélküli hálózatokat építenek. Kiépíthetsz egy tetszőleges méretű zónát, és az a legjobb, hogy a készülékek önbeállítóak, így egy csomópont hozzáadásakor vagy kivételekor nem kell a többi csomópont beállításait módosítanod.
- Az Elan SC520 watchdog timer támogatása. Ez kiváltképpen a Soekris beépített watchdog timerét illeti – ez teszi lehetővé az önműködő újraindítást programhibák esetén. Különösen hasznos akkor, ha az AP olyan helyre van felhelyezve, amelyhez nehezen lehet hozzáférni (például egy nyilvános parkban), vagy ha az AP-t nem tudjuk aktívan felügyelni (mint csaknem minden AP-t). Ezzel és az írásvédett fájlrendszerrel a Pebble rendszer karbantartási igénye csaknem egyenlő a nullával.
- Egy áthidaló tűzfal futtatásának támogatása.

A Pebble jól fut egy 486-os processzoron, és nincs szüksége 32 MB RAM-nál többre, valamint 128 MB-nál nagyobb tárhelyre. Valószínűleg futni fog a sufniban tárolt 486-osodon, de 300 dollárnál kevesebből venni tudsz egy nagyon király, és nagyon kicsi Soekris 4511-20-ast és egy vezeték nélküli kártyát – így semmi perc alatt működőképes lesz az egész. Ha megszállott vagy, akkor vehetsz egy Soekrist, amiben nincs tápegység és 250 dollárnál kevesebbért is építhetsz magadnak egy AP-t.
A Pebble bármelyik Intersil Prism2 vagy Prism2.5 alapú 802.11b kártyával működik, ilyen például a Linksys WPC11, a D-Link DWL-650 vagy a WL100 és a WL200.

Néhány egyszerű beállítással bármely Linux-támogatott 802.11b kártyával is képes lesz ugyanerre.
Ha felkészültél a Pebble indítására, nézd meg a projekt honlapját. Az igazán kicsiben gondolkodók a Pebble helyett megszemlélhetik a WISP-Distet (Wireless ISP Distribution ➲ http://leaf.sourceforge.net). A WISP elképesztően apró: 8 MB-os flash ROM-on is elfér és 16 MB RAM kell hozzá. Ez távolról sem olyan összetett, mint a Pebble (ez amolyan pehelysúlyú AP), és testreszabni sem könnyű.
Saját nyilvános AP-d felállításához egy olyan ISP-re van szükséged, amelyik nem törődik azzal, hogy megosztod a sávszélességedet, továbbá egy AP-re, egy célterületre, egy irányított antennára és némi munkakedvre. Néhány ISP, mint például a New York-i Bway.net, boldogan engedi, hogy megoszd az általad fizetett sávszélességet. Mások, mint például a Time-Warner Cable és az AT&T Broadband, lecsapnak azokra, akik megosztják a sávszélességet. Egy helyi nyilvános Wi-Fi-szervezet segít neked egy megfelelő ISP fellelésében, vagy segít lobbizni az ISP-d felhasználási feltételeinek megváltoztatásában. A Freenetworks.org segít neked a hozzád legközelebb eső nyilvános Wi-Fi-csoport megtalálásában.
Az NYCwireless célja a nyilvános helyek megcélzása, például a parkoké. A helyszín nagyon fontos. Mint ahogy azt Doc is felfedezte, amikor megpróbálta a Tudor City Parkot elérni egy fél háztömbnyi távolságból a magasból, a távolság gondot okozhat. Egy jól irányzott antenna segítheti a szűk helyeken való sugárzást, de egy közeli többirányú (omni) antenna majd' minden esetben túltesz egy messzebb lévő, irányzott antennán. A Bryant Parkot több irányból is szolgálják az omni és területi (irányzott) antennák kombinációi újságosbódék tetejeiről. A City Hall Parkot jobban kiszolgálja egy területi antenna az utca túlsó oldalán lévő üzlet tetejéről. A Verizon remek járdaszegélyi szolgáltatást biztosít egyszerű, telefonfülke tetején lévő omni antennáival.
Az antennák nem olcsók, bár nem is túl drágák. És néha az is megteszi, ha kiteszünk egy AP-t az ablakba. *Ben Hammersly* is így járt el a londoni Kynance Mewsban, és ezzel egy egész utcát lefedett, ideértve két utcai kávéházat is. Az egész csak rajtad múlik.

Kurt Starsinic

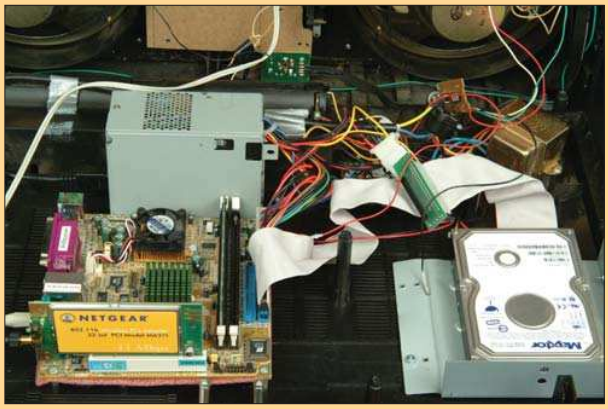
Drazen a Dyne.org-nak is a tagja. Ez egy bécsi központú, szabad programokat használó szakemberekből álló csoport, célja egy GPL felhasználási szerződésű valós idejű videofeldolgozó, médiafolyam- és egyéb nagyszerű dolgot megvalósító program kifejlesztése. Drazen szerint a Dyne egyik legjobb eszköze a HasciiCam, egy egyszerű kis programocska, ami egy tévékártyáról veszi a jeleket, ASCII-jelekké alakítja át őket, majd számos kimenetet képes adni, például frissítő címkés HTML-t, élő ASCII-ablakot vagy egyszerű szövegfájlt.
Drazent leginkább a Dyne:bolic Linux-terjesztés és a MPEG4IP érdekli. A Dyne:bolic egy CD-ről futtatható multimédia-központú változat, ami felismeri a hang-, video-, tévé-, hálózati kártyákat és egyéb eszközöket is. MPEG4IP egy olyan adatfolyamcsomag (streaming), amellyel a kereskedelmi adatfolyamrendszerek használata elkerülhető. Drazen szerint a Dyne:bolic letöltése után fel lehet írni egy CD-re, betölthető róla a Linux, és jó minőségű MPEG4-et tud sugározni.
Drazen úgy gondolja, hogy mindezek a nyílt forráskódot használó erőfeszítések azt fogják eredményezni, hogy szabaddá teszik az audio- és videoszerkesztést, -készítést és -terjesztést, ezeket függetleníteni fogják a vállalatoktól, amelyek igyekezetünk gyümölcseit és a képzeletünket még mindig ellenőrzésük alatt tartják. Úgy tekint a Linuxra, mint nyilvános operációs rendszerre, a Wi-Fi-re pedig mint nyilvános hálózatra. Együtt fogják megalkotni a (szó szerint) nyilvános tévét és rádiót.
A Wi-Fi, a HasciiCam, a digitális videokamerák, az olcsó eszközök, a szabad programok, a Dyne:bolic és az MPEG4IP eredményeképpen Drazen azt várja, hogy a tudósítás és a műsorszórás költsége rendkívüli mértékben esni fog. Amikor ez bekövetkezik, reszkessetek!

Viselhető és hordozható eszközök

Vasárnap visszamentem az Alt.Coffee-ba, hogy találkozzam *Ahmi Wolf*-fal. Ő és *Mark Argo* készítették el a Bass-Stationt, a 80-as évek bőrönd méretű gettó harsonáját, ami egyben digitális jukebox és Wi-Fi hozzáférési pont. Ahmi és Mark

Zenedoboz: a szörny gyomrában

A Bass-Station gyomrában egy 800 MHz-es processzort (➲ http://www.viatech.com) egy mini-ITX alaplapban, 256 MB RAM-ot, egy Prism alapú PCI vezeték nélküli kártyacsatolót és egy 120 GB-os IDE-merevlemezt találunk. Debian Linuxon fut (Woody 3.0), ami a HostAP-meghajtókat használja (➲ http://hostap.epitest.fi), hogy a Wi-Fi kártya hozzáférési pontként szolgáljon, így irányított csomópontként használjunk (managed node), szemben egy egyszerű ügyféllel (client mode). Működik rajta egy DHCP-kiszolgáló is, ami az IP-címeket küldi szét a vezeték nélküli ügyfeleknek. Ez egy általános ISC DHCP-kiszolgáló az alapbeállításokkal használva, amely csaknem az összes Linux-változatnak része. A Bass-Station egy DNS-kiszolgálót is futtat, ami az úgynevezett „pont (.) tartomány" (dot (.) domain) kiszolgálásáért felelős, így az összes tartománykérés a Bass-Station IP-címeire irányítódik.



Vannak más módjai is a felhasználó egy adott weboldalra történő juttatásának. Egy olyan aktív portálprogram, mint a NoCat (➲ http://www.nocat.net) jó erre, de az ilyen programok a hálózat portáljaként vagy belépési pontjaként szolgálnak. Az a gond ezzel a programmal, hogy megpróbálja betölteni a kívánt URL-t, mielőtt a portálra irányítaná a látogatót. Mivel a Bass-Station nincs hozzákapcsolva vagy külső IP-címmel társítva, a program újra és újra megpróbál betölteni valamit, amit nem tud, így hát semmivel sem szolgál – ezért cselhez kell folyamodnunk! A DNS rendszer kialakítását Bind (mi a 9-es változatot használtuk) tiszta telepítésével kell kezdenünk. Ezután a */etc/bind/named.conf*-ban a zónabejegyzést (".") cseréljük a következőkre:

```
zone "." {
type master;
file "/etc/bind/db.root";
notify no;
};
```

Cseréljük fel az alapbeállított *db.root* fájlt (előbb készítsünk róla biztonsági másolatot) egy olyan fájllal, ami a következőket tartalmazza:

```
$TTL 604800
@ IN SOA . root.localhost. (
1 ; Serial
604800 ; Refresh
86400 ; Retry
2419200 ; Expire
604800 ) ; Negative Cache TTL
@ IN NS .
* IN A 192.168.23.1
```

Az 192.168.23.1 IP-címet cseréljük le arra az IP-címre, amelyik tartományokat be akarjuk hozni.
Az adatok egy MySQL-adatbázisban élnek, és egy Apache webkiszolgáló hozza be őket. Ezek együttesen egy mindenféle szolgáltatást ellátó felületet adnak. Egyelőre a következőket tartalmazzák:

- Fájlfeltöltés (HTTP-t használunk, így minden művelet végrehajtható a böngészőben).
- A Bass-Stationön lévő fájlok böngészése, olvasása, illetve letöltése.
- A magnó zenelejátszó szolgáltatásának irányítása.

MPG123-at használunk a médiafájlok lejátszására. Megírtam az MPG123 kiszolgálóoldali irányítóprogramját, ami az adatbázisunkkal is tartja a kapcsolatot. A C++-forrás hamarosan elérhető lesz a honlapunkon.

Ahmi Wolf

4. kép A Bass-Station

5. kép Egyetlen tetőantenna szolgálja ki a City Hall Parkot

6. kép A fekete dudor ezen a nyilvános telefonon egy Wi-Fi-antenna

kivették ennek a régi darabnak a rádiós és a kazettás részét, és különféle korszerű hordozható Wi-Fi- alkatrészeket raktak bele: egy mini-ITX alaplapot, vezeték nélküli kártyát, amit egy antennához kapcsoltak, Debiant (Woody) telepítettek egy CompactFlash memóriakártyára, és egy 120 GB-os merevlemezzel is ellátták. Meghagyták az erősítőt és a hangfalakat, és ezeket az alaplap audiokimenetére csatlakoztatták.

Az eredmény tökéletes ellentéte az iPodnak – mind szociálisan, mind pedig esztétikailag. Egy nagy, csúnya rádiósmagnó, ami egyúttal Linux alapú Wi-Fi-hozzáférési pont és óriási merevlemezzel ellátott jukebox. Az alapötlet az volt, hogy egy mindenféle bulizáshoz alkalmas jukeboxot készítsenek: a parkokban megrendezett partiktól kezdve az egyetemi kollégiumokban való mulatozásig. Mindenki, aki Wi-Fi-n keresztül kapcsolódik a Bass-Stationhöz, hozzájárul a zenéhez és eljátszhatja a lemezlovast, így jutalmazza az együttműködést.

A Bass-Station a szomszédos extraneté, nem egyetlen személyé, és nem is az egész világé. Ahmi ezt a következőképpen magyarázza:

„A Bass-Station nem kapcsolódik más hálózathoz és nem is részese annak. Ő hozza létre a maga kis hálózatát, ami csak a Bass-Station saját hatókörében működik. Egyrészről a Wi-Fi hatóköre korlátozott, de ez a korlátozott hatókör teszi különlegessé. A hálózat használói mind szoros közelségben vannak egymással, ezáltal a közösség részévé válnak – legyen az akár állandó közösség, vagy egy ösztönösen kialakult mobilközösség, mint például a Bass-Station hálózat."

Ahmi bolgár barátja, ***Milena Iossifova***, aki társa a NYU interaktív telekommunikációs projektben, létrehozott egy jó módszert a Wi-Fi kialakítására, ezt WiFisense-nek nevezte el, és úgy hívja, hogy „a vezeték nélküli hálózatok hordozható letapogatója". Ez voltaképpen egy hátizsák 64, három különböző színű LED-del ellátva, amelyek akkor kezdenek el világítani, ha Wi-Fi-működést érzékelnek a különböző csatornákon.

Mindeme derűlátás és energia arra emlékeztetett engem, hogy milyen is volt a Szilícium-völgy a 80-as 90-es években, amikor még nem járta át teljes egészében a korrupció. Ahmi és Milena már letettek valamit ennek az új kultúrának az asztalára.

Egy új háttér kiépítése, egyik követ a másikra rakva

Terry Schmidttel az Emenity irodájában találkoztam, a Wall Streethez közel. Röviden tájékoztatott a nyilvános Wi-Fi kialakításával kapcsolatos kihívásokról, mely Wi-Fit New York egyik különleges városrészében valósítják meg. A NYCwireless és az Emenity első nagy feladata a Bryant Park Wi-Fi-ellátásának kiépítése volt, ami a New York-i közkönyvtárral a középső blokkon osztozik. Terry a következőket mesélte:

„Két általános antennát, egy szektorantennát és két pont–pont kapcsolatot alakítottunk ki magán a parkon belül. Óriási siker volt, így nyilvánvalóvá vált, hogy milyen nagy igény van a nyilvános vezeték nélküli hálózatra. Egy civil szervezet, mint amilyen a NYCwireless is, nem tud egyszerűen szolgáltatási szerződéseket és hasonló dolgokat megkötni – ez az, amit az Emenity tesz.

Terry úgy tekint az Emenityre, mint köztes szervezetre a tisztán önkéntes és teljesen önálló szervezetek között. Például a Bryant Parkot eredetileg a NYCwireless építette meg, majd az Emenity üzemeltette, most pedig a park teljesen önfentartóan működik. Az Emenity legnagyobb vevője a Downtown Alliance, egy vállalkozásfejlesztési körzet (BID-business improvement district), amit abból a célból alapítottak, hogy „megalkossák és reklámozzák a biztonságos, tiszta és teljesen behálózott közösséget". A BID-eket szerte a városban egy kevéske helyi forgalmi adóból működtetik. A Bryant parki fejlesztés – amely rendkívül látványos, figyelembe véve, hogy azelőtt a senkiföldje volt – a BID munkájának eredménye. Mivel a szövetség a tulajdonosokat szolgálja ki, azzal

a kéréssel is megkeresheti őket, hogy ajánlják fel tetőiket vagy ablakaikat a nyilvános helyszínekre irányuló vezeték nélküli antennák felállításához.

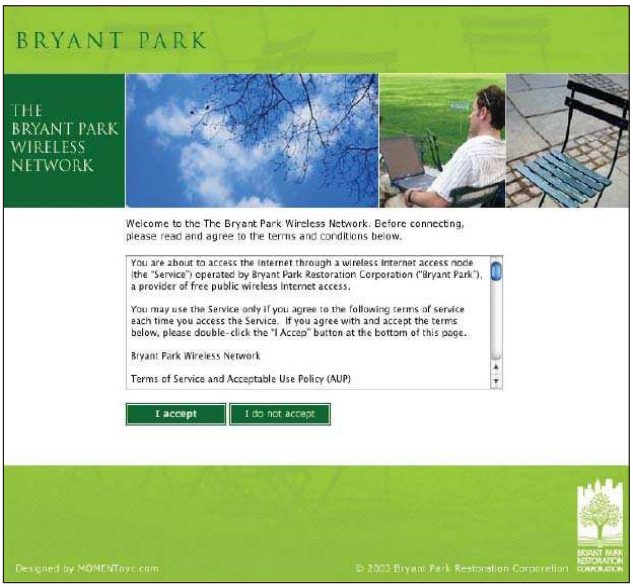
Helyszínek, helyszínek és helyszínek

A City Hall Parknál a szemközti utcában lévő J&R Music and Computer World bolt teteje eszményi helynek bizonyult egy hozzáférési pont elhelyezésére. Egy négyszögletes, fehér antenna került elhelyezésre egy kb. 40°-os irányított szöggel a park irányába. Ez szolgáltatja a park számára a lefedettséget, és még egy kicsit többet is. A park túlsó végén lévő városházánál a jel gyengül. Ez meglehetősen jó elérés, amely egyben más hozzáférési pontokat is kegyesen kiszolgál, a Starbuckét, a városházáét, a Woolworth épületét és más környékbeli helyekét. Terry Schmidt elmondta, hogy a NYCwireless arra biztatja a helyi lakosokat, hogy nyitott hozzáférési pontokat üzemeltessenek „NYCwireless" felirattal, és jegyezzék be magukat a NYCwirelessnél, hogy megjelenhessenek a szervezet végpontlistáján. A végfelhasználói szerződés a zárttól a szabadig széles skálán mozog. A Time-Warner például durván megtagadja a felhasználóktól a sávszélesség megosztásának a jogát. A másik véglet a Verizon, aki a Wi-Fi DSL-vásárlói számára hozzáférési pontokat árusít.

A Verizon tud valamit

A Verizon telefonfülkék ezreivel rendelkezik New York utcáin, és ő is látja ugyanazokat a jeleket a falakon. Egy briliáns ötlettel állt elő: alakítsuk át a telefonfülkéket hozzáférési pontokká! Az első 150-et 2003. május 13-án indították útjára, a vállalat terveiben további ötszáz vagy több városbeli és azon túl lévő fülke ilyen szolgáltatással való ellátása szerepel.
Írásom születésésnek időpontjában a szolgáltatás kizárólag a Verizon üzleti és DSL-ügyfelei számára érhető csak el ingyenesen. De nincs semmi, ami megakadályozhatná a vállalatot abban, hogy más vevők számára is megnyissa a szolgáltatást, vagy hogy teljesen ingyenessé tegye. Így lett megtervezve. Valójában úgy alakították ki, hogy a szolgáltatást a lehető legegyszerűbben hadrendbe lehessen állítani vagy módosíthassák. Ez azért lehetséges, mert a vállalat a Linuxot és a nyílt forrású eszközöket alkalmazza. ***Sean Byrnes***, a Verizon mérnöke ezt a következőképpen világítja meg:
„A Linux lehetővé tette számunkra, hogy szolgáltatásunkat rendkívül gyorsan alakítsuk ki. Így ahelyett, hogy nagy kiszolgálókat telepítettünk volna az egyik adatközpontunkba, Linuxtelepeket építettünk és alapváltozatokat készítettünk, amelyek segítségével rendkívül gyorsan kialakíthattuk a hozzáférési pontokat. A nyílt forráskódú programok széles körét használva sokkal gyorsabban tudtuk ezt megtenni, mintha felhasználási szerződésekre kellett volna várnunk… Nem tudtuk volna a szolgáltatást áttelepíteni egy adatközpontba, ha a Linux nem biztosította volna számunkra a felületfüggetlenséget és azokat a nyílt forráskódú módszereket, amelyek több operációs rendszeren kerültek megvalósításra. Keményen dolgoztunk, hogy a Linux megfeleljen az adatközpontok számára, de ennek még nem jött el az ideje."
Először úgy tűnt számomra, hogy a Verizon egy olyan vállalat, amelyik egyszerűbbnek találta kihozni saját megoldását, mintsem egy beszállító segítségétől függjön. Sean Byrnes szerint „Ez valójában kevesebbet mond a valóságnál." Részletesebben így magyarázza kijelentését:
„Amikor új alkalmazással vagy szolgáltatással jössz elő, tényként lehet elkönyvelni, hogy a legtöbbjük csak ragasztóanyag. Ez még inkább igaz a nagyvállalatok vonatkozásában. Mivel

7. kép A Bryant Parkban ez a pavilon egy hozzáférési pontot tartalmaz



8. kép A Bryant Park nyitólapja

már rengeteg rendszered és alkalmazásod van, össze kell valahogy ragasztanod őket, így hát gyorsnak kell lenni. Sosem arról van szó, hogy képes vagy-e megvenni egy gyártó termékét, majd bevezetni azt az első napon."
Ezzel a sok utcai felügyel hozzáférési ponttal a Verizon emberei értékes és valós tapasztalatokhoz jutnak a Wi-Fi-vel kapcsolatban. Terry Schmidt pesszimista a nem ingyenes Wi-Fi üzleti modellel kapcsolatban: „Nem hisszük, hogy a legtöbb fizetős vezeték nélküli dolognak életképes üzleti modellje lenne. A T-Mobile-hoz hasonló vállalatok – ezekkel a Starbucks helyekkel – mind csak falják a pénzt, és szinte senki sem használja őket."
Mindeközben rengeteg ember veszi hasznát az ingyenes Wi-Fi-nek olyan helyeken, mint a Bryant Park vagy az Alt.Coffee. „Az ingyenes vezeték nélküli hálózat jót tesz az üzletnek" – véli Terry.
Ez a modell. A helyi vállalkozók állásfoglalása a következő: „Üzletemet és a környező piacot azzal teszem vonzóvá, hogy ingyenes vezeték nélküli hálózatot szolgáltatok. Ez csábító

dolog, kiemeli a környezetet és vonzza a vásárlókat."
Úgy működik a Verizon szolgáltatása – ami már létező ügyfelei számára ingyenes –, mint egy virágláda? Úgy vélem, igen. A Verizon egy helyi telefontársaság New Yorkban, sok otthoni és üzleti DSL-felhasználója van. A virágládák, amelyek varázslatszerűen megjelennek e felhasználók számára, ráadást képeznek a jelenlegi szolgáltatáson felül. A Verizon már kijelentheti, hogy „Hozd ki innen a laptopodat, és ülj be egy kávéházba!"

A közélet felélénkítése

A Wi-Fi új és gyakorlatias szolgáltatást nyújt a közélet számára. Az elmúlt két évtizedben a személyi számítástechnika nagy része a falakon belül zajlott, nyomtatókhoz, hálózatokhoz, kiszolgálókhoz és telefonvonalakhoz csatlakozva. Ha a laptopunkat a szabadban használtuk, ez általában ugyanúgy kapcsolat nélkül zajlik, mint ahogy a repülőgépen is történik.
A nyilvános Wi-Fi-vel a világ hálózati tudását a szabadba visszük, és ez megváltoztatja a dolgokat.
Éveken keresztül látogattam a New York-i közkönyvtárat, teljesen figyelmemen kívül esett az a kietlen terület, amit Bryant Parknak hívtak. A legutolsó utam volt az első találkozásom a Bryant Parkkal, mivel az európai kulturális fővárosok nagy parkjainak mintájára teljesen felújították. Gyepével, szökőkútjaival, árnyékos pavilonjaival, a sétányok menti padjaival és a szabadtéri éttermekkel a civilizáció csúcsa volt a szememben. Megszerettette velem a civilizációt és azt a kegyet, ami ezt feljebb emeli. Ez is sokat mond. A közterületek teszik civilizálttá városainkat. Talán ugyanígy fogja civilizálttá tenni a nyilvános Wi-Fi az internetet.

Linux Journal 2003. szeptember, 113. szám

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

KAPCSOLÓDÓ CÍMEK

New York vezeték nélküli hálózatot üzemeltető szervezetei
Downtown Alliance ➲ http://www.downtownny.com
Emenity ➲ http://emenity.com
A New York-i városháza jelentése a városi hálózatok helyzetéről: a széles sávú város építése
➲ http://www.council.nyc.ny.us/pdf_files/reports/broadbandcity.pdf
NYCwireless ➲ http://www.nycwireless.net
A New York Egyetem interaktív telekommunikációs programja ➲ http://www.itp.nyu.edu

Vezeték nélküli hálózatok más városokban
„Kelet felé néző antenna: A Linux és a Wi-Fi Szófiában"
➲ http://www.linuxjournal.com/article/6954
Asheville, Észak-Karolina, Beampost:
➲ http://www.blaserco.com/blogs/2003/02/20.html#a95
Austin, Texas
➲ http://www.austinwireless.net/cgi-bin/index.cgi
Az Intel „legbehálózottabb" városainak listája
➲ http://www.intel.com/products/mobiletechnology/unwiredcities.htm
London ➲ http://www.consume.net
Long Beach, Kalifornia ➲ www.longbeachportals.com
Párizs, Franciaország
➲ http://www.iht.com/articles/95233.html
WAfreenet Perthben
➲ http://www.nodedb.com/australia/wa/perth/?
Portland, Oregon
➲ http://www.personaltelco.net/index.cgi/PersonalTelco
San Francisco, Kalifornia
➲ http://www.bawug.org
Seaside, Kalifornia
➲ http://www.ezgoal.com/hotspots/wireless/f.asp?fid=57748
Seattle, Washington ➲ http://www.seattlewireless.net
Winston-Salem, Észak Karolina
➲ http://www.ezgoal.com/hotspots/wireless/f.asp?fid=65372

Szabad Szoftver Projektek
Bass-Station ➲ http://bass-station.net
Dyne:bolic Linux ➲ http://dynebolic.org
Dyne.org ➲ http://dyne.org
HasciiCam ascii.dyne.org
Kismet ➲ http://www.kismetwireless.net
MPEG4IP ➲ http://mpeg4ip.sourceforge.net
Open Source Streaming Alliance
➲ http://www.streamingalliance.org
Pebble Linux ➲ http://www.nycwireless.net/pebble
WiFisense ➲ http://wifisense.com

Termékek
Lindows MobilePC
➲ http://info.lindows.com/mobilepc/mobilepc.htm
Media Box ➲ http://www.ituner.com/products.htm
Soekris Engineering ➲ http://www.soekris.com
Vezeték nélküli műsorszórás a nyilvános Wi-Fi-n keresztül, Network 2 Cable Network ➲ http://open4all.info/laika

Egyéb
Alt.Coffee ➲ http://www.altdotcoffee.com
Az EFF listája a vezeték nélküli hálózatokat támogató ISP-kről
➲ http://www.eff.org/Infra/Wireless_cellular_radio/wireless_friendly_isp_list.html#list
➲ http://www.linuxjournal.com/article/6955
Jelkutató falfirka ➲ http://www.warchalking.org
Jelkutató utazás
➲ http://www.personaltelco.net/index.cgi/WarDriving
Jelkutató séta
➲ http://www.personaltelco.net/index.cgi/WarWalking

Fedezzük fel a vezeték nélküli hálózatokat!

Nézzünk körül a környéken, és térképezzük fel a meglévő vezeték nélküli összeköttetéseket egy ingyenes alkalmazás és egy Linux alapú tenyérgép a Zaurus segítségével!

Minden jel szerint a vezeték nélküli hálózat terjedésének kellős közepén vagyunk, minden nap egyre újabbak létesülnek országszerte és az egész világon. Mint oly sok szolgáltatás bevezetésének korai szakaszában, úgy tűnik, a szolgáltatóknak most is gondjuk akad egy ilyen értékes szolgáltatás megismertetésével. A vezeték nélküli hálózat felfedezése lakóhelyünkön nagy kihívást jelenthet. Ha városban lakunk, valószínű, hogy a nyilvános vezeték nélküli hálózatok százai csak arra várnak, hogy csatlakozzunk hozzájuk. Míg vidéken talán nehezebb hozzájutni (USA), azért lehetséges. Egy eszközkészlet szükséges hozzá csupán, amivel megtaláljuk és használhatjuk a számunkra nyilvánossá tett hálózatokat.

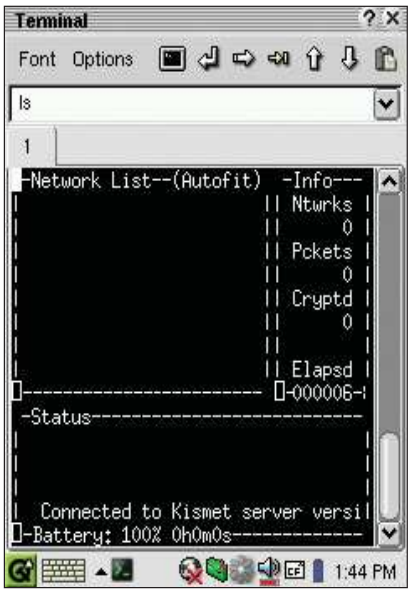
Eszközök

Az első ilyen eszköz maga a számítógép. A lényegében mindenhol jelenlévő, egyre gyorsabb és egyre kisebb laptopok váltak a vezeték nélküli hálózat igazi eszközévé. Használhatunk ezen kívül tenyérgépet (PDA) is. Bár a Zaurust is tenyérgépnek tekintik (vagy a Sharp elnevezése szerint Personal Mobile Toolnak), teljesítménye egy igazi asztali gépének felel meg zsebnyi méretben. A megfelelő programokkal ellátva a Zaurusszal szó szerint sokkal könnyebb a hálózat felfedezése, anélkül, hogy a teljesítményben engedményekre kényszerülnénk.
A Zaurus a hálózatok bejárásához a Kismet nevű programot használja, amely ***.tar.gz*** formátumban a ➲ http://killefiz.de/zaurus címen érhető el. A Kismet használható a Kismet-Qt-vel együtt is, ami egy olyan egy átlátható és rendkívül felhasználóbarát GUI-felület, ami az összes adatot megadja a helyben elérhető hálózatok vizsgálatához és a csatlakozáshoz. A Kismet használatához nincs szükség a Zaurusra, bármelyik Linux-rendszeren működik.

A Kismet parancssoros felületének gyors áttekintéséhez helyezzük be CF-kártyánkat a megfelelő foglalatba, és terminál üzemmódban rendszergazdaként jelentkezzünk be. A hálózati kártya elkezd gyorsan villogni, jelezve, hogy a Kismet az összes beérkező csomagot megvizsgálva vezeték nélküli hálózatok után kutat.

Eszközkészlet

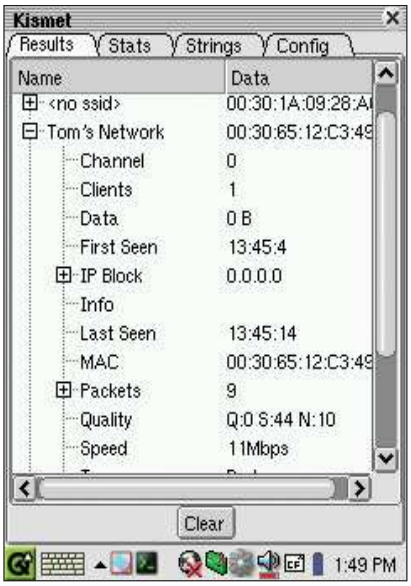
A parancssoros Kismet kijelzi a közelben található hálózatok, valamint a fogadott csomagok számát, és azt, hogy ezekből hány volt titkosított. A parancssoros program testreszabásához a `vi`-vel szerkesszük a ***/home/root/usr/etc/kismet.conf*** fájlt. Ez egy megjegyzésekkel bőségesen tűzdelt beállítóállomány, részletes leírásokkal a Kismet minden egyes beállítási lehetőségéről. Közülük számos használható a Kismet-Qt esetében is.
A Kismet-Qt telepítése valamivel egyszerűbb. A letöltött és a Zaurushoz hozzáadott ***.ipk*** fájl megjelenik az ***Add/Remove Software*** (program hozzáadása/eltávolítása) lehetőségnél a ***Tools*** (eszközök) fülön. Alapesetben ez a fájl a ***/home/zaurus/Documents/Install_Files*** útvonalon található. Az alkalmazást az ***Add/Remove Software*** ablakban a telepítés bejelölésével és a telepítési útvonal megadásával (belső flash vagy külső tárolóegység) választhatjuk ki – ezután a Zaurusra bízhatjuk a munka nehezét. Ha inkább a szöveges alkalmazással szeretnénk telepíteni, váltsunk a fenti könyvtárra, és kövessük az `ipkg` utasításait.
A parancssoros Kismet létrehoz egy kiszolgálót, a Kismet-Qt alkalmazás innen veszi az adatokat. Ha a Kismet kiszolgáló nem fut, a Kismet-Qt hibát jelez, miszerint nem tud a kiszolgálóhoz csatlakozni. A fentiek szerint a kiszolgálót a szöveges ablakból el kell indítanunk, majd ezután megnyithatjuk a Kismet-Qt felületét a Zaurus képernyőjének bal alsó sarkában található ***alkalmazások*** gombra kattintva. A feltelepített Kismet-Qt csomag és



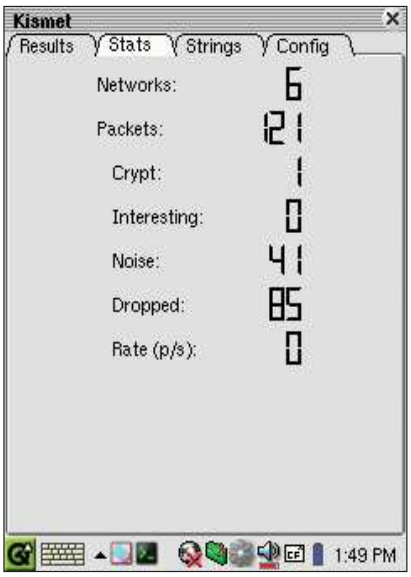
1. kép A Kismet parancssoros felülete

2. kép A Kismet-Qt-felület

a szöveges ablakban futó Kismet segítségével készen állunk arra, hogy felfedezzük a nyilvános vezeték nélküli hálózatokat.



3. kép A Kismet-Qt Results (eredmények) fül

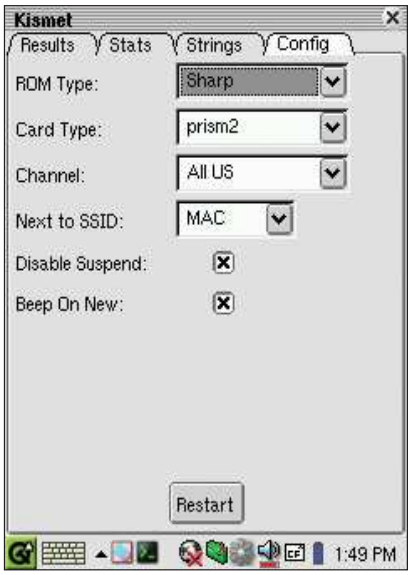


4. kép A Kismet-Qt Stats (statisztikák) fül

A Kismet-Qt felület

Mint azt a *2. kép* mutatja, a Kismet-Qt négy fő részre oszható. Mutatják a pillanatnyi hálózati műveleteket, ezekről egy statisztikát, egy kimutatást az észlelt összes különleges jelsorozatról, valamint egy beállítási képernyőt. Ezek a fülek a legtöbb felhasználó számára bőségesen elegendő adatot szolgáltatnak a vezeték nélküli hálózatok azonosításához és csatlakozáshoz. Vizsgáljunk meg néhány fület külön-külön is.

A Kismet-Qt az éppen futó kapcsolat során észlelt összes hálózati műveletet megjegyzi. Ezek az adatok a ***Results*** (eredmények) fülön jelennek meg, a legfelső szinten a hálózat nevét (vagy ESSID-jét) megmutatva. Bár a ***Results*** fül rengeteg adatot tartalmaz, nem árt egy-két jótanács a hálózatok vizsgálatához. Olyan esetekben, amikor a vezeték nélküli hálózat nem használ ESSID-t, az alapértelmezés szerinti megnevezés szögletes zárójelek között jelenik meg. Ha egy olyan névvel találkozunk, mint például „linksys”, elképzelhető, hogy egy vezeték nélküli magánhálózatba botlottunk, ami a



5. kép A Kismet-Qt Config (beállítások) fül

gyártó eredeti beállításaival működik. Bár lehet, hogy a tulajdonos a szolgáltatását közzé szándékozik tenni, legyünk óvatosak. Ha bármilyen kétely merül fel a hálózat célját illetően, kerüljük a használatát.

A ***Results*** fül lényegében minden adatot a rendelkezésünkre bocsát, ami egy nyilvános hálózathoz való csatlakozás megteremtéséhez nélkülözhetetlen. Nézzük meg ezeket az elemeket részletesebben! Különösen az IP-címtartományt, a csatornát, amin a hálózat működik és az ESSID-t kell megjegyezni ez utóbbit annak eldöntésére, hogy WAP-titkosított-e a jel. Ezek lesznek a kulcsaink a vezeték nélküli birodalomhoz. Oda kell figyelni emellett a jel erősségére és az utoljára észlelt tevékenység idejére („last seen”) is. Ezek mutatják a vezeték nélküli hálózatok felhasználóinak mozgás közben a relatív helyzetüket a csomóponthoz képest. Egy gyengébb jel arra utal, hogy a csomópont szélén vagyunk, akár kifelé, akár befelé mozgunk.

A Kismet-Qt egy jól áttekinthető felületen foglalja össze a pillanatnyi hálózati tevékenységeket. Tartalmazza a tartományon belül jelenleg elérhető vezeték nélküli hálózatok számát, a beérkező és ebből a titkosított csomagok számát, a jelzajszintet és a pillanatnyi, másodpercenkénti nyugtázott csomagok arányát. Mivel ezek a pillanatnyi tevékenységek összegzett értékei, a számok nagyon nagyok is lehetnek, ha egymást átfedő vezeték nélküli zónákkal találkozunk. A fent említett másodpercenkénti arányszám a jelerősséggel és az utolsó aktivitás időértékével együtt utalhat a hálózat középpontjához képesti mozgásunkra.

A ***Config*** (beállítások) fül a Kismet-Qt-ben tartalmazza azokat a legfontosabb adatokat, amelyek a Zaurust hálózatvizslatóvá teszik. Itt kiválaszthatjuk a rendszer ROM-típusát (Sharp vagy OpenZaurus) és a használt CF-kártya típusát. A leggyakrabban használt kártyák szerepelnek a kártyatípus-választási lehetőségek között, de az ***Other*** (egyéb) segítségével a beállításokat kézzel is megváltoztathatjuk, ha kártyánk a listában szereplő protokollok egyikét sem használja. Ezenkívül egy listából a használandó csatornát is kiválaszthatjuk, ami az egyedi, illetve az összes egyesült államokbeli vagy nemzetközi csatornák használatának lehetőségét is tartalmazza. Az utóbbi két beállítás az egyesült államokbeli vagy nemzetközi csatornák teljes tartományának tevékenységét figyeli. A ***Config*** fül talán leghasznosabb lehetősége az, hogy képes hangjelzéssel a tudtunkra adni, ha a program egy új hálózatra bukkan.

A Kismet/Kismet-Qt által szolgáltatott adatok igen szerteágazóak. Az eszköz minden adatot megad a közelünkben hozzáférhető vezeték nélküli hálózatok pásztázásához, csatlakozásához és használatához. A vezeték nélküli hálózatokat használó közösség már itt van a „kertek alján”. A Zaurus, a Kismet valamint a vezeték nélküli CF-kártya segítségével a Linux erejét a szinte bárhol lehetséges csatlakozás lehetőségével tovább növelhetjük.

Linux Journal 2003. szeptember, 113. szám

Tony Steidler-Dennison
(tony@steidler.net)
Az Optical Mechanics üzemeltetési igazgatója. Robotvezérelt csillagvizsgáló-szintű távcsöveket készít, a Linuxot beállítja ezek működtetéséhez, és távcsöveit a világ bármely részén üzembe helyezi.

Bevezetés a pyGTK és a Glade használatába

A pyGTK és a Glade bárki számára lehetővé teszi, hogy könnyen és gyorsan hozzon létre működő grafikus felületeket.

A pyGTK és a Glade szépsége abban rejlik, hogy megnyitották a profi minőségű, rendszerek közötti grafikus programok fejlesztésének útját azok számára, akik nem szeretnének túl elmélyülten foglalkozni a témával, mégis szükségük van egy grafikus felületre a programjukhoz. A pyGTK nemcsak a kezdők számára teszi lehetővé nagyszerű grafikus felületek írását, hanem a gyakorlott programozóknak is, hogy minden eddiginél gyorsabban hozzanak létre rugalmas, változtatható és hatékony felhasználói felületeket. Ha korábban vágytunk már arra, hogy szép megjelenésű felhasználói felületet hozzunk létre gyorsan és minél kisebb erőfeszítéssel, de nem vagyunk jártasak a grafikus felületek írásában, akkor érdemes tovább olvasnunk.

Ez a cikk annak a tanulási folyamatnak az eredménye, amelyen az Immunity CANVAS (➲ http://www.immunitysec.com/CANVAS) program megírása közben keresztülmentem. A grafikus felület fejlesztése közben megtanultak jelentős részét megtalálhatjuk a pyGTK GYK-ban a ➲ http://www.async.com.br/faq/pygtk/index.py?req=index címen. Egy másik leírás, amit minden bizonnyal sokat fogunk használni, ha elmerülünk a pyGTK rejtelmeibe, a ➲ http://www.gnome.org/~james/pygtk-docs címen található. Őszintén kijelenthetjük, hogy egy kisvállalat számára a pyGTK használata versenyképes előnyt jelenthet olyan más grafikus fejlesztőkörnyezetekkel szemben, mint amilyen például a C. Ennek a cikknek az elolvasása után remélhetőleg mindenki képes lesz arra, hogy az összes nyelv közül a legkönnyebben elsajátítható Python segítségével összeüssön egy GUI-t.

Összehasonlításként: a CANVAS grafikus felülete az alapoktól kiindulva körülbelül két hét alatt készült el, anélkül, hogy a pyGTK-ról előzetes ismeretekkel rendelkeztem volna. Ezt követően a GTK v1-ről GTK v2-re egy nap alatt ültettem át (erről később még lesz szó), és most már mind a Microsoft Windows-, mind a Linux-rendszereken működőképes.

A pyGTK rendszereket áthidaló jellege

Egy tökéletes világban soha nem lenne szükségünk arra, hogy a Linuxon kívül bármilyen más rendszerre programot kelljen fejlesztenünk, s ezt kedvenc rendszercsomagunkon tehetnénk meg. A valós világban támogatnunk kell a Linux különböző változatait, a Windowst, a Unixot vagy bármilyen más operációs rendszert, ha az ügyfél úgy kívánja. A grafikus eszközkészlet kiválasztása nem kis mértékben azon múlik, hogy a megbízónk által használt felületen melyik élvez nagyobb támogatást. Napjainkban a Python fejlesztőeszközként történő választása bármilyen új próbálkozásnál már természetessé kezd válni, ha a fejlesztés gyorsasága nagyobb jelentőséggel bír, mint a futtatási sebesség. Ezzel el is érkezünk a Python GUI fejlesztőeszközének választási lehetőségeihez, amit a wxPython, a Tkinter, a pyGTK és a Python/Qt képez.

Szem előtt tartva a tényt, hogy nem vagyok a grafikus felületek fejlesztésének szakértője, érzéseim alapján a következő érvek szólnak a pyGTK választása mellett: a wxPython már hosszú ideje elérhető és vonzó felületek létrehozásának lehetőségét kínálja, de nehéz munkára fogni és használni, különösen egy kezdőnek. Nem beszélve arról, hogy mind a Linux, mind a Windows felhasználói nagyméretű bináris állományt kénytelenek letölteni és telepíteni. A Qt-nek – bár Linux alá ingyenes – Windows alatti használata már jogdíjhoz kötött, s ez a tulajdonsága számos olyan kisvállalkozásnál kizáró tényező lehet, amelyek programjukat mindkét operációs rendszer alatt terjeszteni kívánják.

A Tkinter az első Python GUI-fejlesztőkészlet, szinte minden Linux-terjesztésnek része. Emellett csúnya, és a Tk-nak a Python-programokba történő beágyazását igényli, ami azt az érzést kelti, hogy visszafelé haladunk. Egy kezdő számára nagyon fontos dolog, hogy amennyire csak lehetséges, a grafikus felületet elválassza magától a programtól. Ez lehetővé teszi annak elkerülését, hogy a GUI szerkesztésekor a programon is egy sereg változtatást végre kelljen hajtani, vagy valamilyen módosítást bele kelljen építeni.

A fenti okokból kifolyólag a leginkább a pyGTK az, amire szükségünk van. A pyGTK gondosan különválasztja a programot és a grafikus felületet. A ***libglade*** használatával maga a GUI XML-fájlként tárolódik, amit tovább szerkeszthetünk, többféle változatot készíthetünk belőle, vagy bármit megtehetünk vele, mivel nincs összeépítve a programkóddal. A Glade GUI-készítő programként való használatával gyorsan szerkeszthetünk programfelületeket. A fejlesztési idő olyan rövid, hogyha a különböző felhasználók eltérő felületeket szeretnének maguknak, mindnyájuk igényeit könnyen kielégíthetjük.

A GTK és a pyGTK változatai

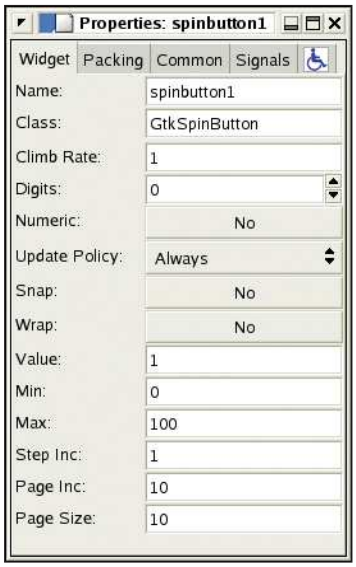
A GTK-nak két változata érhető el, az 1-es és a 2-es, ezért mielőtt GUI-fejlesztő projektünkbe belevágnánk, el kell döntenünk néhány dolgot a fejlesztéssel és karbantartással kapcsolatban. Előfordulhat, hogy le kell töltenünk a Gladev2-változatát, vagy telepítenünk kell a GTK fejlesztői csomagjait ahhoz, hogy a GTK v2 ***libglade*** programkönyvtárat lefordíthassuk. Biztosíthatom olvasóinkat, ez megéri a fáradságot. A GTK v2 számos előnyt kínál: szebb kinézetet, telepítőt a Windowshoz a Python 2.2-es változatával, a hozzáférhetőséggel kapcsolatos bővítményeket, amely vak felhasználók számára teszi lehetővé az alkalmazások elérését. Ráadásul a 2-es változat a frissebb rendszercsomagok közül sokban előre telepítve érkezik, bár még mindig lehetséges, hogy a fejlesztői RPM-csomagokat és a legfrissebb pyGTK-t magunknak kell telepítenünk.

A GTK v2 és így a pyGTK v2 is nyújt néhány, egy kicsit összetettebb elemkészletet. Egy nagy tudású GUI-mester kezében ezek az eszközök csodálatos dolgokra képesek, de a kezdőt csak összezavarják. Igaz ugyan, hogy néhány kódrecept úgy állítja be őket, mintha a GTK v1-es eszközeinek másolataiként – ha egyszer megtanultuk a használatukat – kezelhetőek lennének. Például amikor a GTK v1-es változatával a CANVAS számára a teljes grafikus felületet kifejlesztettem, GTK v2-esben át kellett terveznem (ez pontosan egy napot vett igénybe). Ügyfeleim

linuxos gépeiről hiányzott ugyan a GTK v1 támogatása, de a GTK v2 telepítése egészen egyszerű volt. Az egyetlen kivétel a Ximian Desktop, ami határozottan megkönnyíti a pyGTK és a GTK v1 telepítését. Vagyis ha a teljes ügyfélkörünk ezt használja, esetleg kitarthatunk a GTK v1-es mellett. Egy dolgot azonban jól meg kell jegyeznünk: létezik Python-parancsfájl arra a célra, hogy a Glade v1-esben írt projektünket Glade v2-es változatra alakítsuk át, de fordítva már nem megy. Így ha mindkettőre szükségünk van, először a Glade v1-esben fejlessszünk, ezután alakítsuk át azt és hangoljuk össze a különbségeket.

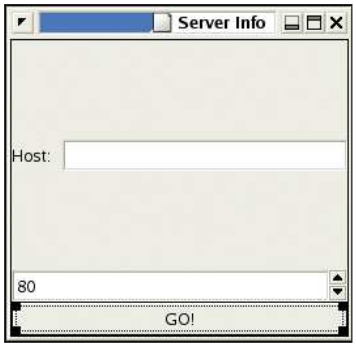
Bevezetés a Glade v2-esbe

A Glade és a ***libglade*** használata mögött az az elgondolás húzódik meg, hogy grafikus felületünket időpocsékolás kódolással létrehoznunk. Csak ülni és mást se csinálni, mint megadni az egyes eszközök helyét, színét és alapértelmezett beállításait a Python-fordítónak – ez bizony igen időrabló tevékenység. Akinek volt már dolga Tcl/Tk-programozással, biztos, hogy napokat töltött el ilyesmivel. És nem csupán ez, de egy kóddal megírt GUI módosítása is tekintélyes vállalkozás lehet a rá fordított időmennyiség tekintetében. A Glade és ***libglade*** esetében a kód írása helyett XML-fájlokat hozunk létre, valamint olyan kódhivatkozásokat, amelyek egy nyomógombot, beviteli mezőt, vagy kimeneti szövegtárolót tartalmazó fájlra mutatnak.
A munka elkezdéséhez – ha még nem rendelkeznénk vele – szükségünk van a Glade v2-esre. Amennyiben már megvan, akkor is jól jöhet a legfrissebb változata. Ha korábban már telepítettük a GTK v2 fejlesztői csomagokat (a `-devel` RPM-csomagokat), a Glade v2 letöltése és telepítése sem okozhat gondot. Mindenesetre a legtöbb, GUI-fejlesztésben még kevéssé jártas felhasználó számára félelmetesen üres a kezdőablak. Programunk fejlesztésének megkezdéséhez kattintsunk a ***Window*** ikonra. Most egy nagy üres ablakot kell látnunk a képernyőn.
Az egyik fontos dolog a grafikus felületek fejlesztésével kapcsolatban az, hogy alapvetően kétféle objektummal dolgozunk: eszközökkel, ilyenek a címkék, a beviteli szövegdobozok és az egyéb látható dolgok, és ezeknek az eszközöknek a tárolóival. A legvalószínűbb, hogy a tárolók három fajtája (a vízszintes doboz, a függőleges doboz és a tábla) közül egyet fogunk használni. Az összetett külső előállításának a legkönnyebb módja az, ha a kívánt sorrendben egymásba ágyazzuk őket. Kattintsunk például a vízszintes doboz ikonjára. A ***window1*** vonalkázott területére kattintva három újabb területet szúrunk be, amelyekre eszközöket helyezhetünk.
Most ennek a három területnek bármelyikét kiválaszthatjuk, és tovább oszthatjuk egy vízszintes dobozzal. Ha nem vagyunk elégedettek az eredménnyel, visszatérhetünk és törölhetjük, kivághatjuk és beilleszthetjük, vagy megváltoztathatjuk a dobozok számát a ***Properties*** (tulajdonságok) menüből (erről még később lesz szó).
Ezekből az elemi alkotórészekből szinte bármilyen elrendezés előállítható. Most, hogy már rendelkezünk egy kiinduló beosztással, olyan eszközökkel tölthetjük fel, amelyeknek már van valamilyen rendeltetésük is. Én most egy címkével, egy szövegbeviteli dobozzal, egy léptetőgombbal (spinbutton) és egy közönséges gombbal töltöm fel. Elsőre nem valami szép látvány. Ne feledjük, hogy a GTK a kész ablakot a megjelenítéskor önműködően méretezi, ezért mindent annyira szorosan helyeztem el, amennyire csak lehetett. Amikor a felhasználó elhúzza az ablak sarkát, az is önműködően változtatja a méretét. Ezeket a beállításokat a ***Properties*** (tulajdonságok) ablakban tehetjük meg (váltsunk a Glade főablakára és kattintsunk a ***View®Show Properties*** menüpontra). Az ablakban a különböző eszközök más-más tulajdonságait változtathatjuk meg. Ha például a léptetőgomb van a fókuszban, akkor az *1. képen* látható lehetőségek állnak rendelkezésünkre.



1. kép
A Glade-ben az eszközök jellemzőinek megváltoztatására szolgáló felület az eszközök függvényében változik

A ***Value*** (érték) tulajdonsággal a léptetőgomb megjelenéskori alapértelmezett értékét módosíthatjuk. Fontos a ***Max*** érték helyes beállítása is. Gyakori hiba a ***Value*** tulajdonság valamilyen magas értékre történő beállítása anélkül, hogy a ***Max*** értéket is megfelelően hozzáigazítanánk. Ennek az az eredménye, hogy a léptetőgomb először a beállított ***Value*** értéket mutatja, de amikor a felhasználó állítani próbálja, visszaáll a ***Max***-értékre – ami meglehetősen zavaró jelenség. Esetünkben ezt az eszközt a TCP-kapu beállítására fogom használni, így 65535-re állítom be, a legkisebb értéket pedig 1-re és az alapértelmezést 80-ra.
Ezután a ***label1*** címkére fókuszálok, és a ***Host:*** feliratot adom neki. A ***window1*** feliratra kattintva a Glade főablakában az egész ablakunk a fókuszba kerül, amelynek így szintén beállíthatjuk a jellemzőit. Ezt úgy is megtehetjük, hogy láthatóvá tesszük az eszközfa (widget tree) ablakát, és ebben kattintunk a ***window1***-re. A nevének ***serverinfo***-ra való változtatásával és a címnek (title) ***Server Info*** értéket adva a programunknak megfelelően állíthatjuk be az ablak fejlécét és az eszköz Glade-beli magas szintű nevét.



2. kép
A Glade-ben a grafikus felület nem teljesen olyan, mint a lefordított, úgyhogy ne aggódjunk a Host: területe miatt

Ha visszatérünk az eszközfanézetre és a ***hbox1***-re kattintunk, növelhetjük a térközt a ***Host:*** felirat és a szövegbeviteli doboz között. Ezzel egy kicsit tovább csinosíthatjuk az elrendezést. Kész grafikus felületünk a *2. képen* látható.
Rendes körülmények közt mindez csak néhány percig tart. Egy kis gyakorlás után látni fogjuk, hogy a Glade használatával még a legösszetettebb GUI létrehozása is csak perceket vesz igénybe. Hasonlítsuk ezt össze azzal az idővel, amit a Tk-parancsok begépelésére fordítanánk, ha ugyanezt akarnánk vele létrehozni.
Ez a felület természetesen még nem csinál semmit. Ahhoz még meg kell írnunk azt a Python-kódot, ami a ***.glade*** fájlt betölti és elvégzi a tényleges műveleteket. Most már valójában két Python-fájlt írok minden Glade által vezérelt projekthez:

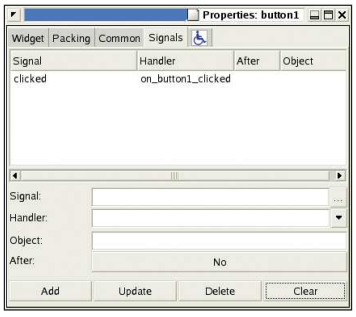
az egyik a GUI-t kezeli, a másik pedig teljesen független ettől a grafikus felülettől. Ezzel a módszerrel egyszerűvé válik a GTK v1-esről a GTK v2-esre vagy akár egy másik GUI-eszközre történő átültetés.

A Python-program létrehozása

Először is a lehetséges változatkülönbségekből adódó gondokat kell elhárítanunk. Én az alábbi kódot használtam, bár a GYK-ban említett bejegyzések némelyike hasonlóan működik:

```
#!/usr/bin/env python

import sys

try:
 import pygtk
  #közöljük a  pyGTK-val, hogy ha lehetséges,
  #a GTKv2-t használnánk
  pygtk.require("2.0")
except:
  #Néhány rendszercsomagban van GTK2,
  #de pyGTK nincs
  pass

try:
  import gtk
  import gtk.glade
except:
  print "You need to install pyGTK or GTKv2 ",
  print "or set your PYTHONPATH correctly."
  print "try: export PYTHONPATH=",
  print "/usr/local/lib/python2.2/site-
        ➥packages/"
  sys.exit(1)

#Most már rendelkezünk beolvasott gtk-val
#és gtk.glade-del,
#Øs az is biztos, hogy a GTK v2 fut
```

Most `appGUI` néven létrehozunk egy GUI-osztályt. Mielőtt ezt megtennénk, meg kell nyitnunk a ***button1*** tulajdonságait, és létre kell hoznunk egy eseményt. Ehhez kattintsunk a három pontra, görgessünk a ***clicked*** feliratra, válasszuk ki, majd kattintsunk az ***Add***-ra.



3. kép
Az eseménykezelő létrehozása után

A *3. képen* látható az eredmény, amit kapnunk kell. Ezzel a gombra való kattintás egy `signal_autoconnect` eseményt hoz létre, ami az egyik függvényünk (`button1_clicked`) hívását eredményezi. A listában látható a többi, kezelésre váró lehetséges esemény is. Minden eszköz különböző várható jelzésekkel rendelkezhet. Például egy szövegváltozási jelzés hasznos lehet egy szövegbeviteli dobozon, de egy gombon értelmetlen, mert annak a felirata nem szerkeszthető.
A program elindításával és a `gtk.mainloop()` futtatásával működésbe lép a szerkezet. A különböző eseménykezelőknek eltérő számú értékekkel kell rendelkezniük. A kattintás eseménykezelője csak egyetlen értékkel bír: annak az eszköznek a nevével, amin a kattintás történt. Amíg a főablakon vagyunk, adjuk hozzá a ***destroy*** (megsemmisít) eseményt, így ha ezt az ablakot bezárjuk, a program futása véget ér. Ne feledkezzünk meg Glade projektünk mentéséről sem!

```
Glade project.
class appgui:
  def __init__(self):
    """
    Ebben az initben a Server Info
    főablakát jelenítjük meg
    """
    gladefile="project1.glade"
    windowname="serverinfo"
    self.wTree=gtk.glade.XML
(gladefile,windowname)
    # csak két regisztrálandó visszahívásunk
    # van, de akárhányat regisztrálhatnánk,
    # vagy használhatunk egy különleges
    # osztályt a visszahívások önműködő
    # regisztrálására, ha egy értéket
    # szeretnénk átadni, az alábbi tuple-hoz
    # hasonlót használnánk (??):
    # dic = { "on button1_clicked" :
    # (self.button1_clicked, arg1,arg2) , ...


dic = { "on_button1_clicked" :
             self.button1_clicked,
             "on_serverinfo_destroy" :
             (gtk.mainquit) }
    self.wTree.signal_autoconnect (dic)
    return

#####visszahívások
  def button1_clicked(self,widget):
    print "button clicked"

# Így indítjuk el a programot...
app=appgui()
gtk.mainloop()
```

Ha a pyGTK telepítését forráskódból végeztük, nagyon fontos, hogy a `PYTHONPATH` környezeti változó a ***/usr/local/lib/python2.2/site-packages/*** helyre mutasson, hogy a rendszer megtalálja a pyGTK-t. Ne mulasszuk el a ***project1.glade*** fájlnak a pillanatnyi könyvtárunkba való másolását sem. A ***GO!*** feliratra kattintáskor egy csinos üzenetnek kell megjelennie a terminálablakban.
Ahhoz, hogy a program valami érdemlegeset is csináljon, valahogy meg kell adnunk, hogy melyik gépet és kaput használjuk. A következő kódrészlet a `button1_clicked()` függvénybe másolva elvégzi ezt a feladatot:

```
host=self.wTree.get_widget("entry1").get_text()
port=int(self.wTree.get_widget(
  "spinbutton1").get_value())
if host=="":
  return
import urllib
page=urllib.urlopen(
```

```
  "http://"+host+":"+str(port)+"/")
data=page.read()
print data
```

Ha most a *GO!*-ra kattintunk, programunknak el kell látogatnia egy távoli honlapra, be kell olvasnia az oldal tartalmát, és ki kell azt listáznia a terminálablakra. Tovább finomíthatjuk az eredményt azáltal, hogy még több sort hozunk létre a `hbox`-ban, és egyéb eszközöket – például menüsort – adunk a programhoz. Próbálkozhatunk azzal is, hogy `table` elemet használunk az egymásba ágyazott `hbox`-ok és `vbox`-ok helyett, ami az elemek egymáshoz való igazításával gyakran szebb elrendezéshez vezet.

A TextView eszköz

Talán mégsem arra vágyunk, hogy az összes szöveg a terminálablakban jelenjen meg, nem igaz? Valószínűbb, hogy egy másik eszközön vagy ablakon szeretnénk látni. A GTK v2 erre a célra a `TextView` és `TextBuffer` eszközöket használja. A GTK v1 egy könnyen érthető eszközt kínál, amit egyszerűen GTKText-nek neveznek. Adjunk egy ***TextView***-t a projektünkhöz, és az eredményt írassuk ki ebbe az ablakba. Látni fogjuk, hogy egy ***scrolledwindow*** elem jön létre, és arra vár, hogy beépítsük. A ***TextBuffer*** létrehozásához a lenti sorokat adjuk hozzá `init()` függvényünkhöz és fűzzük hozzá ***TextView*** elemünkhöz. A GTK v2 használatának egyik nyilvánvaló előnye, hogy ugyanazt az átmenetitár-tartalmat két különböző módon is megnézhetjük. Ha akarjuk, a ***scrolledwindow1 Properties*** ablakában módosíthatjuk az ablak tulajdonságait, nagyíthatjuk például a méretét, hogy megfelelő hely álljon rendelkezésre a tartalom megjelenítéséhez:

4. kép
A GO!-ra kattintva betöltődik a weboldal és megjelenik a TextView-ban

```
self.logwindowview=self.wTree.get_widget
➥("textview1")
self.logwindow=gtk.TextBuffer(None)
self.logwindowview.set_buffer(self.logwindow)
```

A `button1_clicked()` függvényünkben a `print` utasítást cseréljük ki az alábbi sorokra:

```
self.logwindow.insert_at_cursor(data,len(data))
```

Most már akárhányszor a *GO!*-ra kattintunk, az eredmény az ablakunkban jelenik meg. A főablakunkat vízszintes panelekkel felosztva, ha kedvünk tartja, ezt az ablakot átméretezhetjük (*4. kép*).

A TreeView és a List eszközök

A GTK v1-es változattal ellentétben a GTK v2-esben a fa (tree) és a lista (list) alapvetően ugyanazt jelenti, a különbség a használt tárolási módban van. Egy másik fontos fogalom a `TreeIter`, ami egy lista vagy fa bizonyos sorára mutató pointer tárolására használt adattípus. Önmagában semmilyen hasznos eljárást nem kínál, azaz a `++` műveletet nem használhatjuk arra, hogy a fa vagy lista elemein lépegessünk. Ellenben ha bármikor egy fa bizonyos elemére akarunk hivatkozni, a `TreeView` eljárásai ezt az értéket kapják meg. Erre látunk példát a következő kódrészletben:

```
import gobject
self.treeview=[2]self.wTree.get_widget
➥("treeview1")
self.treemodel=gtk.TreeStore
➥(gobject.TYPE_STRING, gobject.TYPE_STRING)
self.treeview.set_model(self.treemodel)
```

Ezzel egy kétoszlopos famodellt határoztunk meg, amelynek az oszlopai egy-egy karakterláncot tartalmaznak. A következő kód címet ad az oszlopoknak:

```
self.treeview.set_headers_visible(gtk.TRUE)
renderer=gtk.CellRendererText()
column=gtk.TreeViewColumn
➥("Name",renderer, text=0)
column.set_resizable(gtk.TRUE)
self.treeview.append_column(column)
renderer=gtk.CellRendererText()

column=gtk.TreeViewColumn
➥("Description",renderer,text=1)
column.set_resizable(gtk.TRUE)
self.treeview.append_column(column)
self.treeview.show()
```

A következő függvényt arra használhatjuk, hogy kézzel vihessünk adatokat a faszerkezetbe:

```
def insert_row(model,parent,
➥firstcolumn,secondcolumn):
    myiter=model.insert_after(parent,None)
    model.set_value(myiter,0,firstcolumn)
    model.set_value(myiter,1,secondcolumn)
    return myiter
```

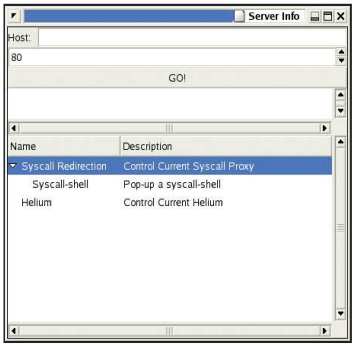
Íme egy példa a függvény használatára. Ne felejtsük el a ***treeview1***-et hozzáadni a glade-fájlunkhoz, majd menteni és átmásolni a helyi könyvtárunkba.

```
model=self.treemodel
insert_row(model,None,´Helium´,´Control
➥Current Helium´)
syscallIter=insert_row(model,None,´Syscall
➥Redirection´,´Control Current
➥Syscall Proxy´)
insert_row(model,syscallIter,
➥´Syscall-shell´,´Pop-up a syscall-shell´)
```

A *5. képen* látható pillanatkép mutatja az eredményt. Mint látható, a `TextView`-t egy `TreeView`-val helyettesítettem. A lista hasonló módon kezelhető, azzal a különbséggel, hogy `TreeStore` helyett `ListStore`-t használunk, és az `insert_after()` függvény helyett nagy valószínűséggel a `ListStore.append()` függvényt.

Párbeszédablakok használata

A párbeszédablak egy egyszerű ablaktól a leginkább abban különbözik, hogy valamilyen értéket kell visszaadnia. Egy párbeszédablak létrehozásához kattintsunk a ***dialog box*** gombra, majd adjunk nevet az alkotóelemnek. Ezután a kódunkban a `[3]gtk.glade.XML(glade-file,`*`párbeszédablak_neve`*`)` függvénnyel adjuk át. Ezt követően hívjuk meg a `get_widget(`*`párbeszédablak_neve`*`)` függvényt, hogy azonosítót rendeljünk ehhez az eszközhöz, majd hívjuk a `run()` eljárását. Ha az eredmény `gtk.RESPONSE_OK`, akkor a felhasználó az ***OK***-ra kattintott.
Ha nem, akkor vagy a ***Cancel***-t választotta, vagy bezárta az ablakot. Ebben az esetben a `destroy()` függvénnyel az ablakot megsemmisítjük és eltüntetjük.
A párbeszédablakok használatának egyik hátulütője, hogyha kivétel történik az elem `destroy()` eljárásának meghívása előtt, akkor egy semmire nem válaszoló párbeszédablak maradhat a képernyőn, ami igen zavaró lehet a felhasználó számára. Figyeljünk arra, hogy közvetlenül az adott válasz és az összes, a párbeszédablaktól várt adat beérkezése után hívjuk meg az eszköz `destroy()` eljárását.



5. kép
Példa a TreeView-ra két oszloppal

Az input_add() és a gtk.mainiteration() használata a foglalatok kezelésére

Egy nap talán éppen olyan pyGTK-programot írunk, ami foglalatokat (sockets) használ. Ha így lenne, vigyázzunk arra, hogy amíg az események kezelése folyamatban van, a programunk semmi mással nem foglalkozik. Amikor például egy `socket.accept()` eljárás végrehajtására várunk, zavaró lehet egy semmire nem válaszoló programot nézni. Ehelyett használjuk inkább a `gtk.input_add()` függvényt a szükséges számú foglalalt létrehozására, amelyek az eseményeket beolvassák a GTK belső listájára. Ez lehetővé teszi, hogy visszahívásokat adjunk meg, amelyek a foglalatokon keresztül érkező bármilyen adatot kezelni képesek.
Az ebben a módszerben rejlő egyik csapda az, hogy az esemény folyamán gyakran frissíteni akarjuk az ablakainkat, és ez szükségessé teszi a `gtk.mainiteration()` meghívását. Viszont ha akkor hívjuk meg a `gtk.mainiteration()` függvényt, amikor az éppen végrehajtás alatt áll, a program lefagy. A CANVAS esetén ezt én úgy oldottam meg, hogy a `gtk.mainiteration()` minden hívását egy olyan vizsgálattal vettem körül, ami biztosította, hogy nem történik rekurzív függvényhívás. Ellenőrzöm a folyamatban lévő eseményeket – ilyen a `socket.accept()` – minden olyan esetben, amikor naplóbejegyzést készítek. Naplózó függvényem a következőképpen fejeződik be:

```
def log(self,message,color):
"""
a naplózóablakra naplózza az eseményt most
éppen nem veszi figyelembe a szín értékét
"""
        message=message+"\n"
        self.logwindow.insert_at_cursor
        ➥(message,len(message))
        self.handlerdepth+=1
        if self.handlerdepth==1 and
        ➥gtk.events_pending():
            gtk.mainiteration()
        self.handlerdepth-=1
        return
```

Egy GUI-nak GTK v1-esből GTK v2-esbe történő átírása

A pyGTK GYK-nak az a része, ami a programok GTK v1-esből GTK v2-esbe való átültetéséről szól, egyre jobban közelít a befejezéshez. Ennek ellenére óvatosaknak kell lennünk azok miatt a nehézségek miatt, amikkel esetleg szembetalálkozhatunk. Az kézenfekvő, hogy minden `GtkText` elemet `Gtk.TextView` elemmel kell helyettesítenünk. Az ehhez kapcsolódó kódot a GUI-ban szintén meg kell változtatnunk, hogy hozzáigazítsuk a végrehajtott cseréhez. Ehhez hasonlóan az összes listát vagy fát, amit a GTK v1-esben létrehoztunk, újra létre kell hoznunk. Meglepetésként érhet minket, hogy a párbeszédablakokkal kapcsolatos teendőket is újra el kell végeznünk, hogy a GTK v2-es sokkal szebben mutató formátumára alakítsuk át őket.
Néhány olyan szövegbeli változtatást is el kell végeznünk, mint a GDK ***gtk.gdk***-ra, vagy a ***libglade gtk.glade***-re történő cseréje. A legtöbb esetben ez egy egyszerű keresés–csere művelettel elvégezhető. Továbbá a `GtkText.insert_defaults` használatos a `GtkTextBuffer.insert_at_cursor()` helyett és a `radiobutton.get_active()` a `radiobutton.active` helyett. A Glade v1-es fájlunkat Glade v2-essé alakíthatjuk a ***libglade*** csomagjának Python-parancsfájljával. Ez a grafikus felület tekintetében megteszi az első lépéseket, de előfordulhat, hogy be kell töltenünk a Glade v2-est, és újra elvégeznünk még néhány beállítást, még mielőtt a kódot átültetnénk.

Befejező megjegyzések

- Ne felejtsük el, hogy a kivágás–beillesztés műveleteket használhatjuk a Glade eszközfáján. Ez felgyorsítja és megkönnyíti az áttervezést.
- Oldjuk fel (unset) a lehetséges elhelyezkedéseket a ***Properties*** ablakban, hogy az indulókép ne nézzen ki olyan furcsán.
- Ha olyan kérdésünk van, amiről azt gondoljuk, hogy más is szembekerülhet vele, vegyük fel a pyGTK GYK-ba. A Gnome IRC-kiszolgálón működik egy hasznos ***#pygtk*** csatorna. Nem lettem volna képes a CANVAS megírására a csatornán keresztül kapott segítség nélkül, amelyek közül különösen ***John Henstridge***-é volt értékes számomra. Nagy elismerése a nyílt forrás közösségének, hogy a kezdők gondjainak megoldása érdekében a legfontosabb fejlesztők is gyakran elérhetők. A kidolgozott bemutató kód megtalálható az 52. CD Magazin/pyGTK könyvtárában, illetve letölthető a ➲ ftp.ssc.com/pub/lj/listings/issue113/6586.tgz címről.

Linux Journal 2003. augusztus, 113. szám

Dave Aitel
A New Yorkban működő Immunity, Inc biztonsági tanácsadó társaság alapítója. A CANVAS az Immunity betöréstesztelő és -felderítő fejlesztői keretprogramja, ami teljes egészében Pythonban, a pyGTK használatával készült.

Titkosított saját könyvtárak megvalósítása

Használjuk ki a visszacsatoló eszközön keresztül titkosított fájlrendszerek előnyeit és a pam_mountot adataink erősebb védelme érdekében.

Amennyiben megfelelően használjuk a titkosított fájlrendszereket, kitűnő eszközt kapunk, amellyel a számítógépünkön található érzékeny adatokat elrejthetjük a kíváncsi tekintetek elől. Az általános titkosító eszközök, például a GNU Privacy Guard (GPG), tökéletesek arra, hogy elektronikus leveleinket titkosítsuk. De a használatuk kényelmetlen olyan fájlok esetében, amelyeket nap mint nap gyakorta módosítani szeretnénk. A GPG-vel ellentétben a titkosított fájlrendszer a felhasználó számára teljesen láthatatlan, ugyanis nincs szükség a fájl használat előtti visszakódolására, se az ezt követő újbóli titkosításra. Ilyen módon a felhasználói feledékenység kérdése is megoldódik. Miután bemutatunk néhány, Linuxon használatos titkosított fájlrendszert, megláthatjuk, hogyan hozhatunk létre olyan titkosított saját könyvtárakat, amelyek a felhasználó belépésekor fűződnek be, kilépéskor pedig önműködően leoldódnak. Végül pedig kiderül, milyen hátulütői lehetnek annak, ha a különféle titkosítási módszereket nem megfelelően használjuk. Miért lehet szükség arra, hogy valaki titkosítsa a számítógépén található adatokat? Hiszen a fájlok jogosultságai is pont erre szolgálnak, nem? Mindamellett, hogy a felhasználói jogosultságok hasznosak, értelmüket veszítik olyankor, ha egy támadó sikeresen átveszi a rendszer és a hozzá tartozó tárolóeszközök feletti irányítást, ilyen módon kerülve meg a jogosultsági rendszert. Tegyük fel, ha valaki ellopja a Linux-rendszert futtató Apple iBook hordozható számítógépemet – ekkor a fájlrendszerbeli jogosultságok értelmetlenné válnak, mivel a tolvaj akár egy saját CD-ről is újraindíthatja a rendszert. Ugyanez áll fenn, ha a laptopomat visszaküldöm az Apple-höz javításra, és egy megbízhatatlan munkatárs szándékosan belenéz a fájljaimba. Emiatt a számítógépen található fájlok titkosítása megfelelő védelemnek tűnhet, mivel a titkosítási folyamat nem függ az operációs rendszer épségétől, fájljainkat a titkosítás védi.
Az iBookomon csak a saját könyvtárakat kódolom le. A teljes fájlrendszernek a gyökértől kezdődően történő titkosítása az én esetemben a fellépő teljesítménycsökkenés és egyéb okok miatt nem járható út. Amennyiben mégis egy ilyesfajta titkosítást szeretnénk megvalósítani, az interneten bőven találhatunk leírást e témáról; az eljárás a Linux rendszerindító memórialemez (ramdisk) képességeire épül. Az a tapasztalatom, hogy x86-os rendszer esetén az általam választott titkosítási módszer körülbelül ötven százalékkal lassabb, mint egy titkosítatlan XFS fájlrendszer, ha a lemezre íráskor átmeneti tárazott (puffered) bevitelt, illetve kimenetet alkalmazunk. Ha pusztán a saját könyvtárakat titkosítjuk, a rendszerben lévő egyéb fájlok, mint például a rendszernapló, egyszerű szövegként tárolódnak, de az én esetemben ez nem minősül igazán érzékeny adatnak. Számomra elfogadható megoldásnak tűnik csak a saját könyvtárak titkosítása.
A Linux kínál néhány lehetőséget a fájlrendszerek titkosításához. Az olyan rendszerek, mint a Transparent Cryptographic File System (TCFS, vagyis átlátszó titkosított fájlrendszer), egy titkosító bővítményt (plugin) ad az NFS által kiszolgált `ext2`-es kötetekhez. Némely fájlrendszer beépített titkosítási lehetőséget is tartalmaz helyi használatra. Esetemben a legjobb megoldásnak a visszacsatoló (loopback) eszközön keresztüli titkosítás mutatkozott. Mint látni fogjuk, a visszacsatoló eszközön keresztüli titkosítással bármilyen, Linux által támogatott fájlrendszert titkosíthatunk, legyen az `ext2`, ReiserFS vagy XFS. A visszacsatoló eszköz segítségével Linux alatt fájlokat fűzhetünk be a rendszerbe olyan módon, mintha a fájl egy eszköz lenne, például a */dev/hda1*. Ez a szolgáltatás arra használható, hogy például CD-re történő égetés előtt a kiírandó ISO fájlt befűzzük a rendszerbe, és ellenőrizzük, valóban működőképes-e, illetve a segítségével megtekinthető a hajlékonylemez-lenyomatok (floppy lemezkép) tartalma is. ***Herbert Valerio Riedel*** GNU/Linux CryptoAPI rendszermagfoltjával és `util-linux` foltjaival rendszerünket – a visszacsatoló eszközön keresztül – titkosított kötetek (lemezrészek) befűzésére tehetjük képessé. Mielőtt mélyebben beleásnánk magunkat a részletekbe, nézzük meg, hogyan fűzhetünk be köteteket a visszacsatoló eszközön keresztül az eredeti rendszermagba. Először is hozzunk létre egy fájlrendszert tartalmazó fájlt, azaz a példában szereplő állományt, ami elég nagy ahhoz, hogy egy 20 megabájtos fájlrendszer elférjen benne.

```
dd if=/dev/zero of=simaszoveg.img bs=1M
➥count=20
```

Ezt követően fűzzük be a fájlt a visszacsatoló eszközhöz:

```
losetup /dev/loop0 simaszoveg.img
```

Majd pedig hozzunk létre egy fájlrendszert a befűzött fájlban:

```
mkfs -t ext2 /dev/loop0
```

Végül fűzzük be a fájlrendszert, ahogy azt egy egyszerű eszköz esetében tennénk:

```
mount /dev/loop0 csatolási_pont
```

Most nézzük meg, hogyan működik mindez titkosított fájlrendszerek esetében. Ehhez azonban a rendszermagnak tartalmaznia kell a hozzá kapcsolódó bővítményt, de sajnos a legtöbb linuxos terjesztés ezt alapesetben nem támogatja, így egy saját építésű rendszermagra lehet szükségünk. Ezt a titkosító felületet a 2.4-es rendszermagokhoz a ➲ http://www.kerneli.org címen érhetjük el, a 2.5-ös rendszermagok viszont már eleve tartalmazzák ezt a bővítést. Ha sikerült feltenned a `patch-int` és `loop-hvr` foltot, a rendszermagod beállítópanelje egy ***Cryptographic options*** (Titkosítási beállítások) almenüvel bővül; ebben a következő beállításokat kell engedélyezned:

- cryptographic API support (`CONFIG_CRYPTO`)
- generic loop cryptographic filter (`CONFIG_CRYPTOLOOP`)
- cryptographic ciphers (`CONFIG_CIPHERS`)

A CONFIG_CIPHER_ részből legalább egy titkosítást engedélyezned kell. Én az AES-t választottam (ezt eredetileg Rijndaelnek hívták), és a példákban is ezt fogom használni.
Ezt követően fordítsd le a rendszermagot a kívánt beállításokkal. A titkosítási részek modulként is fordíthatók. Ha modulokat használsz, mielőtt ténylegesen használatba vennéd, ne felejtsd el betölteni őket. Ne felejtsd el az util-linux foltjait is feltenni, majd fordítsd és telepítsd fel. Az ehhez kapcsolódó util-linux folt a ➲ http://www.kernel.org/pub/linux/kernel/people/hvr/util-linux-patch-int címről tölthető le. Ha sikerül feltelepítened az új változatokat, tapasztalni fogod, hogy a mount és losetup parancs működése némileg megváltozik. Most már készen állunk arra, hogy titkosított fájlrendszer hozzunk létre, a korábbi példában látottakhoz hasonlóan, amikor egy egyszerű visszacsatoló eszközön keresztül fűztük be a fájlrendszert. Először is, hogy nehezen felismerhetővé tegyük a szabad és a foglalt területeket, az állomány tartalmát a */dev/urandom* segítségével véletlen adatokkal töltjük fel, a */dev/zero* helyett.

```
dd if=/dev/urandom of=ciphertext.img
➥bs=1M count=20
```

Miután létrehoztuk a gazdafájlt, először ideiglenesen hozzá kell kapcsolnunk egy visszacsatoló eszközhöz, csakúgy, mint az előbb. Ezúttal azonban meg kell mondanunk a losetup-nak, hogy az eszköz titkosítva legyen, esetünkben AES kódolással:

```
losetup -e aes /dev/loop0 ciphertext.img
```

Ha a losetup kérdezi, írd be a szükséges jelszót, és ha kéri – a kódolástól függően – a kulcsméretet, amellyel az adott kötet titkosítva lesz. A fájlrendszer létrehozása ugyanúgy történik, mint ahogyan az előző példában is láthattuk. A titkosítást már beállítottuk, és a visszacsatoló eszközre bíztuk, így azzal már nem kell foglalkoznunk:

```
mkfs -t ext2 /dev/loop0
```

A losetup módosításán kívül a util-linux folt a mount parancsot is felkészíti a titkosított kötetek kezelésére. Ezáltal a titkosított kötetek befűzése meglehetősen egyszerűen történik:

```
mount -o loop,encryption=aes ciphertext.img
➥csatlakoztatási pont
```

Ezután a mount parancs megkérdezi a jelszót, és ha szükséges, a kulcsméretet is.
Most pedig, hogy már magadtól is kezelni tudod a titkosított köteteket, rátérünk a pam_mount működésére. A pam_mount egy PAM-bővítmény, ami még jobban leegyszerűsíti a kötetek kezelését, és csak akkor fűzi be a szükséges kötetet, ha a felhasználó bejelentkezik a rendszerbe. Segítségével befűzhetjük a Samba vagy Novell alapú megosztásokat is, akár titkosított fájlrendszereket is. ***Elvis Pfützenreuter*** a pam_mount eredeti szerzője; ***Mukesh Agrawal*** pedig azt a foltot írta, amely először tette lehetővé a titkosított kötetek kezelését is. Jelen pillanatban e cikk szerzője a pam_mount karbantartója, mely bővítmény a ➲ http://www.flyn.org címen érhető el.
Ahelyett, hogy a titkosított köteteket magunknak kellene befűznünk, a rendszergazda a pam_mount-ot beállíthatja oly módon, hogy az magától fűzze be a fájlrendszert, ha a felhasználó bejelentkezik, illetve leoldja, amikor a felhasználó kilép a rendszerből.

Ez úgy állítható be, hogy a belépési jelszó egyúttal a titkosított kötetet is elérhetővé teszi, így hozva létre egy teljesen átlátszó titkosított fájlrendszert.
A pam_mount három különféle módszert használ arra, hogy hozzáférjen a titkosított kötetekhez. Az első elég unalmas. Ha a titkosított kötet jelszava nem áll kapcsolatban a belépési jelszóval, akkor a helyes jelszót a felhasználótól kéri be. E módszer használatához a ***pam_mount.so***-t és a pmhelper-t kell helyesen beállítani. A szokásos ./configure, make, és make install parancsok segítségével a pam_mount és a beállításokat tartalmazó fájlok a helyükre másolódnak. A gyári ***pam_mount.conf*** a */etc/security* könyvtárban található. Vess rá egy pillantást, és állítsd be a te rendszerednek megfelelően – a beállításfájlban található temérdek megjegyzésnek köszönhetően ez nem okozhat gondot. A legfontosabb változtatás, hogy a kötetek befűzéséhez kapcsolódó hivatkozásokat hozzáadjuk a fájl végéhez. Ahogyan a beállításokat tartalmazó fájlban látható, a titkosított köteteket kezelő beállítások körülbelül így festenek:

```
volume user local ignored
befűzendő_eszköz
csatlakoztatási_pont mount_kapcsolói
kódolás_típusa
kulcs_elérési_útja
```

Alább egy példát láthatunk arra, hogyan fűzzünk be egy AES-szel kódolt fájlrendszert a */home/mike* alá, amikor Mike bejelentkezik a rendszerbe:

```
volume mike local - /home/mike.img /home/mike
➥loop,user,exec,encryption=aes,keybits=256 - -
```

Ezt követően a megfelelő PAM-beállításfájlhoz adjuk hozzá a következő sorokat:

```
auth required pam_mount.so try_first_pass
session required pam_mount.so try_first_pass
```

Ez a beállításfájl ahhoz a szolgáltatáshoz tartozzon, amelyikhez a titkosított kötetkezelést szeretnéd kötni. Noteszgépemen a */etc/pam.d/login* fájl így néz ki:

```
auth      requisite  pam_securetty.so
auth      requisite  pam_nologin.so
auth      required   pam_env.so
auth      required   pam_unix.so nullok
account   required   pam_access.so
account   required   pam_unix.so
session   required   pam_unix.so
session   optional   pam_lastlog.so
session   optional   pam_motd.so
session   optional   pam_mail.so standard noenv
password  required   pam_unix.so nullok
                     ➥obscure min=4 max=8 md5
auth      required   pam_mount.so try_first_pass
session   required   pam_mount.so try_first_pass
```

Végül pedig hozd létre a felhasználóhoz tartozó titkosított kötetet, úgy, ahogy az előbb a példában láthattuk.
A kötetek befűzésére a második pam_mount módszer a felhasználó számára egy kicsit kényelmesebb. Ha ugyanúgy hozzuk létre a titkosított fájlrendszert, mint az előző módszer esetében, akkor a titkosított kötet jelszava megegyezhet a

felhasználó belépési jelszavával is, így a pam_mount ugyanezzel a jelszóval fér hozzá a kötethez.
A harmadik példa a legrugalmasabb, és egyúttal ez igényli a legrészletesebb magyarázatot. Először is tisztázzunk néhány fogalmat, hogy megértsük, pontosan hogyan működik ez a módszer:

- sk: a rendszerjelszó. Ez az a kulcs vagy jelszó, amellyel a felhasználó belép a rendszerbe.
- fsk: a fájlrendszer jelszava. Ez az a kulcs, amivel a pam_mount lehetővé teszi, hogy a titkosított kötetet befűzd.
- E és D: egy OpenSSL alapú kódoló, illetve dekódoló módszer, ilyen például a bf-ecb.
- efsk: a titkosított fájlrendszer kulcsa, efsk=E_sk(fsk), ami valahol a fájlrendszerben található (például */home/fhnév.key*).

A pam_mount beolvassa az efsk-t a fájlrendszerről, végrehajtja az fsk = D_sk(efsk) műveletet, és az fsk segítségével befűzi a fájlrendszert. Ennek a módszernek az az előnye, hogy olyan módon is megváltoztathatjuk a belépési jelszavunkat, hogy a titkosított fájlrendszert nem kell újból létrehoznunk. Ha a belépési jelszó megváltozik, egyszerűen csak újból létre kell hozni az efsk-t (vagy a */home/fhnév.key* fájlt) az efsk = E_newsk(D_oldsk(efsk)) segítségével – a pam_mount-ban található egy ***passwdehd*** nevű parancsfájl ennek kezelésére.
A harmadik módszernek a kivitelezéséhez először hozzunk létre egy fájlt, ami a fájlrendszert fogja tárolni (csakúgy, mint az előző példákban):

```
dd if=/dev/urandom of=/home/user.img bs=1M
➥count=méret_megabájtokban
```

Ezután hozzunk létre egy fájlt (efsk), ami a kötet jelszavát tartalmazza (fsk) a */dev/urandom* segítségével, ez a felhasználó bejelentkezési jelszavának megfelelően van kódolva:

```
dd if=/dev/urandom bs=1c count=keysize / 8
➥| openssl enc -bf-ecb >/home/user.key
```

Ezután hozzunk létre egy titkosított fájlrendszert a visszacsatoló eszközön. A fájlrendszer jelszava fsk legyen (ezt a */dev/urandom*-ból nyertük, és a */home/fhnév.key* fájlban a második lépésnek megfelelően titkosítva tároljuk):

```
openssl enc -d -bf-ecb -in /home/fhnév.key
➥| losetup  -e aes -k keysize -p0
➥/dev/loop0 /home/user.img
mkfs -t ext2 /dev/loop0
umount /dev/loop0
losetup -d /dev/loop0
```

Végül a ***pam_mount.conf***-ban állítsd be a kódolás típusát arra, amivel a kulcsfájlt titkosítottad, ez esetünkben a bf-ecb eljárás, majd a kulcs elérési útját állítsd be az efsk elérési útjára, ami esetünkben a */home/fhnév.key* állomány.
Bruce Schneier alapműnek számító könyvében, az Applied Cryptographyben azt állítja, hogy a „programalapú titkosítás ijesztő". Ez alatt azt érti, hogy rendkívül nehéz igazán megbízható titkosítási eljárásokat olyan általános célú operációs rendszereken létrehozni, mint amilyen a Linux is. Például az ilyen korszerű operációs rendszerek a memória tartalmát bármikor kiírhatják a lemezre, és így titkosított és titkosítatlan jelszavak egyaránt a lemezre kerülhetnek. Egy titkosított kötet lényegét veszti, ha a használatához szükséges kulcsot az operációs rendszer kiírja a lemezre. Egy lehetséges módszer ennek elkerülésére az, ha titkosítjuk a cserememéjának használt lemezrészünket. Ennek biztonságos végrehajtására még a CryptoAPI sem képes, de a fejlesztés már zajlik. Egy hasonló kezdeményezés – a LoopAES – azonban már képes titkosítani a rendszer csereterületeit.
Vegyük újra azt a példát, amelyikben a meghibásodott iBookomat elküldtem az Apple-nek javításra. Ebben az esetben bár a saját könyvtáram titkosítva van, az adataim még sincsenek teljes biztonságban. Egy megbízhatatlan munkatárs elindíthatja ördögi CD-ROM-ját a gépemen, és így kicserélheti mondjuk a login programomat az ő saját tervezésű változatára. Így ha visszakapom a noteszgépemet, és bejelentkezem rá, akkor az átjavított login program elküldheti a jelszavamat egy távoli számítógépre. Egy betörésérzékelő rendszer minden bizonnyal kiszúrná ezt a módosítást.
Egy további gyenge pontja a pam_mount-tal befűzött titkosított köteteknek a rendszer bejelentkezési jelszava. Mivel adatainkat a bejelentkezési jelszó segítségével titkosítjuk – akár közvetlenül, akár közvetve –, ennek a jelszónak erősnek kell lennie.
Ahelyett, hogy vakon növelni kezdenénk a szükséges jelszavak minimális hosszát, pillantsunk inkább bele Bruce Schneier „Titkok és hazugságok" című könyvébe. Az az erős jelszó, amit leírva egy pénztárcában tárolnak, biztonságosabb, mint egy fejben tartott, de könnyen kitalálható jelszó. Valamilyen fizikailag azonosító módszer alkalmazása is megfontolandó. Emlékezzünk, ha a rendszered jelszava nem biztonságos, akkor a titkosított fájlrendszered is csak annyit ér, mint a jelszó.
A titkosított jelszavak kétélű kardként is működhetnek. Mi történik, ha elfelejted a jelszavadat? Mi lesz, ha a harmadik megoldást használod, és véletlenül letörlöd a titkosított kulcsodhoz tartozó fájlt? Mi van akkor, ha a programom vagy valaki másnak a titkosító programja hibás? A felsoroltak mindegyike azt eredményezheti, hogy 2128-féle vagy még több különböző titkosított kulcsot kell kipróbálnod, hogy visszakapd a fájlrendszeredet. Az adataidról mindig gondoskodnod kell: állandóan készíts róluk mentést, akár egy jó rendszergazda. Alapesetben ezeket a mentéseket nem kell titkosítani, fizikailag azonban jól el kell zárni valamilyen biztonságos helyre.
Egy szó, mint száz, rengeteg erőfeszítésbe kerül, hogy egy valamennyire is biztonságos rendszert karbantartsunk, azon túl, hogy adataink biztonságát valamilyen korszerű titkosító eljárás szavatolja, például az AES. ***Matt Blaze*** a következőket fűzte hozzá az Applied Cryptography című könyvhöz:
„A kiváló minőségű titkosító eljárások és protokollok nagyon fontos eszközök, de magukban csak a valóság gyenge utánzatai – rendkívül fontos meggondolni azt, hogy mit védünk valójában, és jó tudni, hogy a védelmi rendszerünk hogyan válhat működésképtelenné (a behatolók ritkán tartják magukat a tiszta, jól meghatározott, tanított támadási modellekhez)."
Átrágtad magadat az írásomon, most már tehát tisztában kell lenned a titkosított fájlrendszerek működésével. A megfelelően titkosított saját könyvtárakkal ugyanis nagyon megbízható védelemhez jutunk.

Linux Journal 2003. augusztus, 112. szám

Mike Petullo (lj@flyn.org)
Az amerikai hadsereg Németországban állomásozó szakaszvezetője. Nappal repülőgépekből ugrál ki, éjszaka C-kódokkal hadakozik, és a Linuxot javítgatja 1997 óta.

Eseménykezelő alrendszer Linuxhoz

A távközlési alkalmazások rendkívül alacsony késleltetést követelnek meg, és a bonyolultságuk szintén nem mindennapi. Egy aszinkron eseménykezelő rendszerre építkezve teljesíthetők az elvárások.

Ez az írás egy sorozat első része, amely egy új linuxos eseménykezelő alrendszert ismertet. Ez alkalommal a motivációt és az elvárásokat tárgyalom, illetve egy ilyen távközlési szintű Linux megvalósításának az előnyeit ecsetelem. Egy natív eseményalapú rendszer Linux-rendszermagon belüli megvalósítási lehetőségének tanulmányozása kutatási tervezetként 2001-ben kezdődött meg az Open Systems Labnél (az Ericsson Research egyik egysége), Montrealban. A cél az volt, hogy a Linuxot eseményvezérelt környezettel bővítsék, amely távközlési alkalmazásokban a meglévő megoldásoknál jobb teljesítmény tudna nyújtani.

A Linux operációs rendszer távközlési szintűvé történt fejlesztése iránt egyre nagyobb érdeklődés mutatkozott. Például az OSDL Carrier-Grade Linux munkacsoport (➲ http://www.osdl.org/projects/cgl) jelenleg is azon dolgozik, hogy kidolgozza mindazokat a követelményeket, amelyeket teljesítve a Linux a következő nemzedékbeli hálózatok megbízható, távközlési szintű kiszolgálórendszerévé válhat.

A távközlési alkalmazásoknál az operációs rendszernek rendkívül alacsony válaszidőt kell garantálnia, miközben a leállási ideje éves szinten legfeljebb öt perc lehet – ez 99,999 százalékos rendelkezésre állásnak felel meg –, és ebbe a vas és az operációs rendszer meghibásodásai és a programfrissítések egyaránt beletartoznak. Egy távközlési rendszer esetében ezek mellett további elvárásoknak is meg kell felelni, például a méretezhetőség, az elérhetőség és a teljesítmény tekintetében.

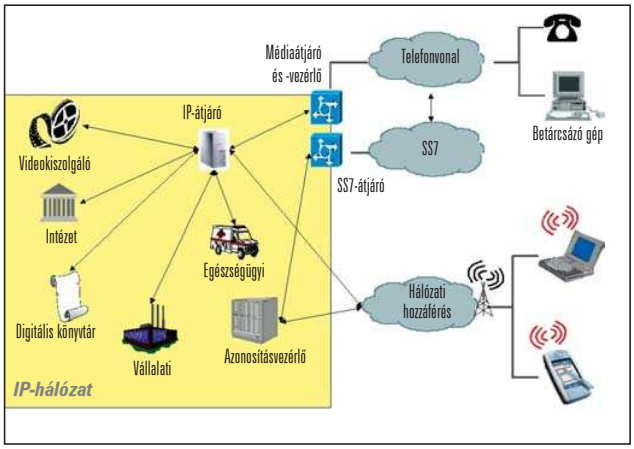
Az ilyen rendszereknek egyszerre több ezer kérést kell kezelniük olyan módon, hogy a teljesítményük még erős terhelés mellett sem csökkenhet. Az előfizetők egy-egy kérés kiadásakor elviselnek ugyan némi késleltetést, ám a féktelenül növekedő válaszidők számukra egyértelműen elfogadhatatlanok. Bizonyos tranzakciók kezelése több okból sem történik meg azonnal, a válaszra néhány milliszekundumnyi ideig vagy néhány másodpercig várni kell. Amíg azonban egy alkalmazás valamilyen válaszra vár, kevésbé tud foglalkozni a többi tranzakció feldolgozásával.

Számos, a Linux képességeinek javítását ígérő megoldás látott napvilágot, köztük többszálas megvalósításokat is találni lehetett, amelyeknek hatékony Posix-felületek készítésétől vagy a meglévő rendszermageljárások méretezhetőségének javításától remélték az áttörést. Mi úgy gondoljuk, hogy e megoldások egyike sem felel meg kellő mértékben a távközlési szintű kiszolgálók igényeinek.

Ahhoz, hogy eme álláspontunk megfelelő alátámasztást nyerjen, ebben a cikkben át szeretném tekinteni a távközlési hálózatok működését. Célom az, hogy ilyen módon tisztázzam a távközlési szintű operációs rendszerekkel szemben támasztott elvárásokat. Ennyi bevezetés után elmagyarázom egy natív aszinkron eseménykezelő rendszer előnyeit – ezek kivétel nélkül a távközlési iparág igényeinek magasabb szintű kielégítését szolgálják.

A távközlési ipar elvárásai

Távközlés, telekommunikáció – esetünkben ez azt jelenti, hogy két eszköz között létrejön egy telefonhívás, és vezetékes összeköttetésen hangot továbbítunk. Az *1. ábrán* a hagyományos



1. ábra A PSTN és az IP alapú hálózatok felépítése és együttműködése

nyilvános kapcsolt telefonhálózatok (Public Switched Telephone Network, PSTN) vázlata látható. A szolgáltatók eszközei a jelzéskezelésnek nevezett művelethalmazzal kezelik a két pont között felépülő telefonhívásokat. A PSTN-hálózatokban a jelzéskezelés a Signaling System 7 (SS7) protokollkészlettel történik. Az SS7 végzi a hívások útválasztását, ennek során – a megfelelő áramkörökön keresztül – egy útvonal jön létre a célkészülék felé. Az útvonal létrejötte után a két telefonkészülék összekapcsolódik, és megkezdődhet a hangtovábbítás. Az SS7 protokoll képes a hívások útválasztására és átirányítására, illetve bizonyos hibákat is kezel.

Az internet rugalmassága, költséghatékonysága és folyamatos növekedése számos távközlési céget arra késztet, hogy internetes alapokra helyezze át szolgáltatásait. Ez is hozzájárul ahhoz, hogy az IP alapú megoldások lassan a távközlési szolgáltatások új szabványaivá válnak. A két különböző típusú hálózat eltérő műszaki megoldásokat alkalmaz, együttműködésükhöz jelzéskezelésre és médiaátjárókra van szükség.

Az átjárók végzik az adatok fordítását a két eltérő típusú hálózat között. Egy SS7 átjáró például a jelzéseket a Stream Control Transmission Protocol (SCTP) segítségével ágyazza be, így teszi lehetővé IP alapú továbbításukat. A médiaátjáró feladata a PSTN-hálózat és az IP alapú hálózat között átadott hangok kódolása és dekódolása.

A jelzéskezelés és a médiaátjáró együttesen képes arra, hogy a PSTN-hálózatból érkező hívások számára elérhetővé tegye az IP alapú hálózatot. A jelzéskezelő átjáróknak beágyazást kell végezniük, megőrizve az SS7 üzenetek írásmódját és szemantikáját,

miközben azok továbbítása internetes protokollok segítségével, például IP felett SCTP vagy IP felett UDP megoldással történik. Egy jelzéskezelő kiszolgálónak az egy időben érkező kérések számának növekedésekor is változatlan teljesítményt kell nyújtania. Egy médiaátjáró segítségével a szolgáltatók megoldhatják az adatfolyamok átléptetését a PSTN és az IP alapú hálózatok között. Az, hogy hány kapcsolatot képesek fogadni, az adott eszköztől függ, a kapcsolatok száma felületenként egy és több ezer között változhat. A médiaátjáróknak nagyszámú kapcsolatot kell kezelniük, méghozzá valós időben.

A hitelesítő, engedélyező és számlázó (authentication, authorization, accounting – AAA) kiszolgálók felhasználói profilokból tartanak fenn adatbázisokat. Általában egy vagy két AAA kiszolgáló egy-egy szolgáltató sok millió előfizetőjének adatait kezeli. Nem szokatlan, hogy csúcsidőben másodpercenként több ezer hitelesítési kérés is érkezik. A kapcsolatok számának szeszélyes változása miatt meglehetősen nehéz előre tervezni. Az AAA kiszolgálók kritikus szerepet játszanak az IP alapú hálózathoz való hozzáférés vezérlésében, így esetükben a meghibásodások nem engedhetők meg. Nem szigorú valós idejű képességekkel kell rendelkezniük, válaszidejük a kérések túlnyomó részében nem haladhatja meg a néhány msec időtartamot.

A médiakiszolgálók különleges erőforrásokat és szolgáltatásokat nyújtanak a felhasználók számára, például videokonferencia és elektronikus levelezési lehetőséget, videokiszolgálókat és egyéb alkalmazásokat. Az ilyen távközlési rendszereknél fontos szempont a méretezhetőség, továbbá a processzorok vagy a felületek számának növelésével, illetve a sávszélesség bővítésével a kezelhető tranzakciók számának is lineárisan kell növekednie. A távközlési szolgáltatók mindig lineáris méretezhetőségben gondolkodnak, vagyis az egy tranzakcióra eső költség a kiszolgáló bővítésekor nem növekedhet.

Meghibásodás vagy terven kívüli leállás esetén a távközlési szintű berendezések önműködően helyreállítják magukat, illetve a redundáns hálózati erőforrások felhasználásával más kiszolgálóknak adják át a feladataikat. Az üzem közben végzett programfrissítések és az üzem közben cserélhető alkatrészek szintén a 99,999 százalékos rendelkezésre állást szolgálják.

A Linux már bizonyítottan üzembiztos, kiegyensúlyozott teljesítményre képes, így a távközlési szolgáltatók számára is figyelemre érdemes megoldást jelent. Ahhoz azonban, hogy a távközlési rendszerek alapelemévé válhasson, olyan összetevőkkel kell bővíteni, amelyek révén képes teljesíteni az iparág rendkívül szigorú kívánalmait.

Teljesítmény és módosuló felépítés: párban járnak

A hagyományos programozási modelleknél a programok összetevői explicit módon végzik egymással az összehangolást. Ha sok adatcserét kell végezni, ez a megszokott módszer. A fájlleírók változásainak figyelésére például a `select()` vagy a `poll()` hívást szokták használni. A `select` általános megvalósítása a leírók egész tömbjét végigolvassa. Méretezhetőségről ez esetben nem is beszélhetünk, hiszen egy-egy leíró módosulásának felismerése a tömb méretével arányosan növekvő ideig tart. Ez az alkalmazások késleltetéseinek növekedését okozza, illetve a rendszer általános teljesítményének romlását idézi elő. Ha átvizsgálunk egy leírótömböt, vagy valamilyen adatra várakozunk, akkor fogyasztjuk a processzor erőforrásait. A hatékony algoritmusok tervezésekor általánosan elterjedt ötlet a rendszeresemények aszinkron kezelése. A felhasználói térben futó alkalmazások eseményekről való értesítésére alkalmas megoldás többek közt a Posix AIO, az epoll vagy a BSD kqueue (lásd a *Kapcsolódó címeket*).

Amikor az ilyen módszerek hatékonyságát vizsgálják, sokszor azt az átlagos időt mérik, amelyik az eseménynek a rendszermag által történő észlelése és az alkalmazás által való tényleges kezelése között telik el. Az az oka, hogy ilyen jellegű vizsgálatokat végeznek, hogy a mikro-teljesítménypróbák ezekben az esetekben hamis eredményt adnának. Ezek a megoldások önmagukban lehetnek ugyan hatékonyak, ám más az alkalmazási területnek kevésbé megfelelő, például többszálú eljárásokkal társítva már kevésbé teljesítenek jól. Számos webkiszolgáló például egy szálkészletet (thread) alkalmaz, amely a kiszolgáló indításakor jön létre. Általában egy szál kezeli a bejövő kapcsolatokat, és mellette minden tranzakcióhoz létrejön egy-egy szál. Alacsonyabb számú bejövő kapcsolatnál ez a megközelítés megfelelő, ám amikor a terhelés magasabbra szökik, a hatékonysága jelentősen leromlik.

Többszálú alkalmazásokat akkor kell használni, ha magas fokon egyidejű futtatást kell megvalósítani a processzorért versengő objektumok között. Jól ismert példát szolgáltatnak a nagyteljesítményű, számításokat végző alkalmazások, amelyeknél fontos, hogy minden szál gyorsan fusson, ám viszonylag kisszámú szállal kell megbirkózni.

A szálak soros és szinkronműködést valósítanak meg, az egyidejű futtatást igénylő alkalmazásoknál gyakorlatilag szabványos megoldássá váltak. Csakhogy az alkalmazások tervezésekor vagy az összehangolások kezelésekor elkövetett hibák miatt könnyen torlódások keletkezhetnek a rendszerben, amelynek a teljesítménye látványosan csökkenhet. ***J. Ousterhout*** „Why Threads Are a Bad Idea" (Miért rossz ötlet a szálak használata?) című írásában megállapítja, hogy a szálak programozása nehéz, és a használatuk általában olyan alkalmazások létrejöttéhez vezet, amelyek nagyobb terhelés alatt meglehetősen rosszul teljesítenek.

A távközlési alkalmazásokban nincs versengés a szálak között. Egyidejűség akkor fordul elő, ha közös objektumokat, például osztott adatszerkezeteket kell kezelni. Ezeknél az alkalmazásoknál a szálak az osztott adatokhoz való egyidejű hozzáférés lehetőségének biztosításához szükségesek.

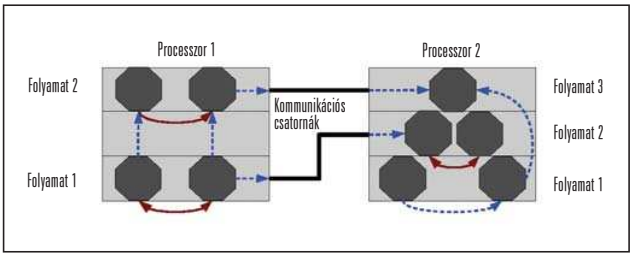
A távközlési alkalmazásoknál másodpercenként több ezer tranzakciót és több száz egyidejű kapcsolatot kell kezelni ugyanazzal a processzorral. Ezek mellett rendszereseményeket, például adatbázis-hozzáféréseket, alkalmazáshibákat, túlterhelésjelzéseket, riasztásokat, a rendszerösszetevők állapotának változásait is kezelni kell. Egy alkalmazás futását egy rendszeren több ezer esemény létrejötte kísérheti, vagyis az eseményeknek szálakkal való kezelése nem volna hatékony.

A hagyományos aszinkron módszerek úgy próbálják kezelni ezt a méretezhetőségbeli hiányosságot, hogy megkísérlik megakadályozni az alkalmazásokat abban, hogy feleslegesen várakozzanak, illetve a linuxos epollhoz hasonlóan segítik a működő leírók hatékony felismerését. Ezeknek a módszereknek a hatóköre sajnos a fájlleírókra korlátozódik, így a figyelembe veendő eseményeknek csak a töredékét képesek kezelni. Nagyszámú szálat indítva – mint az események kezelése miatt az a webkiszolgálók esetében is történik – torlódás alakulhat ki a rendszerben, a helyzet pedig tovább súlyosbodhat.

Eseményalapú rendszerek

Az összetett, osztott programrendszerek fejlesztéséhez olyan eljárásokat kell megvalósítani, amelyek képesek a rendszer rendelkezésre álló erőforrásainak maradéktalan kiaknázására. Ígéretes ötletnek tűnik, ha a kérdés megoldására egy eseményalapú alrendszert próbálunk meg Linux alá készíteni. Egy ilyen bővítménnyel valódi együttműködést lehetne megvalósítani az

operációs rendszer és az alkalmazások között. Bizonyos összetevők segítségével jelentkezni lehetne a különféle eseményekre, amelyekről később aszinkron módon, a megfelelő kezelők végrehajtása révén lehetne értesülni.
Ha összehasonlítjuk a jelzés- és eseménykezelőket, az utóbbiak sokkal hasznosabbaknak bizonyulnak, hiszen az adatokat azonnal át is adják az alkalmazásnak. Egy aszinkron eseménykezelő rendszert alapvetően általános felhasználói kezelők – amelyeket a rendszer hív meg –, illetve periodikus figyelő összetevők, például időmérők készítéséhez lehet felhasználni.
Az első eset különösen érdekes, ha egy alkalmazás nem tudja, hogy mikor történik meg egy esemény. Ha az eseményeket aszinkron módon fogadja, az alkalmazás olyan módon is elvégezheti a szükséges műveleteket, hogy nem foglalkozik az adatok beszerzésével – ugyanis átadott értékek formájában kapja meg őket.
Folytattak már vizsgálatokat arra vonatkozólag, hogyan lehetne létrehozni egy gyors üzenetátadó rendszert, amely az aszinkron eseményeknél megismert alapelvek alapján működne. Például az aktív üzenetek (lásd a *Kapcsolódó címeket*) aszinkron módon kerülnek végrehajtásra, a fogadó folyamat vermében. A felbukkanó szálaknál minden kezelőhöz külön futtatási szál jön létre, az egyszálú felhívásoknál pedig minden egyes processzoron kinevezett szál keletkezik. Az AEM olyan rohamosan fejlődő megoldás, amely natív környezetet ad a ténylegesen aszinkron működést megkövetelő alkalmazások fejlesztéséhez. Az AEM felhasználásával például natív aszinkron foglalatfelületet készítettünk TCP-protokollhoz. AEM környezetben a bejegyzés végrehajtásakor kell kiválasztani a kezelőt, ennek futtatása vagy a pillanatnyi folyamaton belül, vagy új szálként történik meg. Léteznek más kutatási tervezetek is, amelyek hasonló elgondolások alapján próbálják a nagy terhelésnek kitett webkiszolgálók viselkedését javítani (a *Kapcsolódó címek* között lásd az „A Scalable and Explicit Event Delivery Mechanism for UNIX” című írást).
Az eseményközpontú szemlélet legfontosabb előnye az, hogy az események és a szálak kezelését egyetlen alrendszerrel oldja meg. Ebből következik, hogy teljes ellenőrzést enged az erőforrások felhasználása felett.
Az eseményalapú rendszereknél mindig a teljesítmény javítása a cél. Ha az eseménykezelést leválasztjuk az alkalmazásról, az a különféle memóriafoglalási minták alkalmazása vagy az ütemező döntéseinek befolyásolása révén magasabb fokú lokalitást tesz lehetővé. A nem szigorúan valós idejű válaszadásokat például olyan módon lehet megvalósítani, hogy a folyamatok fontosságát (priority) a várakozó események száma határozza meg.
Az egyre nagyobb teret nyerő szemlélet a többszálú rendszerek bonyolultságához képest sokkal egyszerűbb és természetesebb programozási stílust teremt. Hatékonysága látványosan megmutatkozik a többrétegű programrendszerek fejlesztésekor, amelyeknél az egyes rétegek a felsőbb szintűek számára nyújtanak szolgáltatásokat. Az ilyen jellegű rendszerek az osztott alkalmazások területén teljesen megszokottak.
A 2. *ábrán* egy jellegzetes elosztott, eseményvezérelt modellre épülő alkalmazás működése látható. Több programösszetevőből áll, és az alkalmazás minden rétegét egy folyamat valósítja meg. Az osztott alkalmazásoknál azonos és eltérő szintek között, távolra és helyben egyaránt jelentős adatforgalommal kell számolni.
Ezeknek az alkalmazásoknak sokszor olyan szolgáltatásokat kell nyújtaniuk, amelyek a világ bármely pontján kiváló teljesítménnyel üzemeltethetők. Alapvető, hogy az alkalmazások kihasználják a rendelkezésre álló eszközerőforrásokat, hogy a rendszer



2. ábra Egy többrétegű, osztott alkalmazás, amit két processzoron futó, eseményalapú modell feldolgozásával terveztek. Minden réteg egyetlen szálból áll, az alkalmazás összetevői közötti adatcsere szinkron (folytonos vonalak) és aszinkron (szaggatott vonalak) lehet

képességeit tekintve lineáris méreteződésre legyenek képesek. Már a programok tervezésekor gondoskodni kell arról, hogy holtpontok vagy versenyhelyzetek ne alakulhassanak ki az összetevők között. Az ilyen jellegű tervezési hibák végzetes hatást gyakorolnának a rendszerre. Többszálas megközelítést alkalmazva nehéz az ilyen helyzeteket elhárítani, a nagyszámú lehetséges élethelyzet miatt ugyanis túlságosan bajos felismerni és feloldani őket. Eseményalapú megközelítést alkalmazva kevesebb meghibásodási pont kerül a rendszerbe, mivel az aszinkron módon indított szálak számát folyamatosan kézben tartjuk. Könnyebben biztosítható a kezelők futtatásának osztatlansága is, hiszen az alrendszer a rendszermag részét képezi.
A gépek erőforrásai korlátozottak, az elindítható folyamatok száma ugyancsak véges. Bejegyzéskor mód nyílik a futtatni kívánt kezelő kiválasztására. Így olyan alkalmazások készíthetők, amelyek a terhelés növekedésekor is üzembiztosak maradnak. Az alkalmazások szempontjából elsősorban a soros és az aszinkron kód keverésének lehetőségét érdemes kiemelni.
Ha ezt sikerül elérni, akkor olyan alkalmazásokat lehet tervezni, amelyek képesek mindkét szemlélet jó tulajdonságainak a kihasználására.

KAPCSOLÓDÓ CÍMEK

G. Banga és mások „A Scalable and Explicit Event Delivery Mechanism for UNIX”, USENIX Annual Technical Conference, 1999. június, 253–265. oldal
R. Bhoedjang és mások „Friendly and Efficient Message Handling”, 29. Annual Hawaii International Conference on System Sciences, Hawaii, USA, 1996. január, 121–130. oldal
J. Lemon „Kqueue: A Generic and Scalable Event Notification Facility”, FreeBSD Project.
D. Libenzi „/dev/epoll” ➲ http://www.xmailserver.org/linux-patches/nio-improve.html
J. Ousterhout „Why Threads Are a Bad Idea”, USENIX Technical Conference, 1996. január 25.
T. von Eicken és mások „Active Messages: A Mechanism for Integrated Communication and Computation” 19. International Symposium on Computer Architecture, Ausztrália, 1992. május, 256–266. oldal
Linux AEM honlap ➲ http://aem.sourceforge.net
The Open Group, „Posix Asynchronous I/O”, The Single UNIX Specification ➲ http://www.opengroup.org
OSDL CGL (Open Source Development Lab – Carrier-Grade Linux) ➲ http://www.osdl.org/projects/cgl

Eseményalapú keretrendszert alkalmazva a szolgáltatók a szolgáltatást csak a legkisebb mértékben zavarva végezhetnek dinamikus erőforrás-átcsoportosítást. Az eszközök cseréjét és a programok frissítését üzem közben, a rendszer újraindítása nélkül kell elvégezni. Az osztott alkalmazások nagyszámú, egymással párbeszédet folytató összetevőből épülnek fel, az ilyen programok frissítése pedig roppant kényes művelet. A távközlési rendszereknél az összes szolgáltatás rendelkezésre állásának el kell érnie a 99,999 százalékot. A szolgáltatások nem állíthatók le a karbantartások idejére, hiszen ez más szolgáltatói rendszerek működését is befolyásolná, az előfizetőkről nem is beszélve. A programfrissítéseket lépcsőzetesen kell elvégezni. Az eseményalapú alrendszerek lehetővé teszik, hogy az osztott alkalmazásokat ilyen képességekkel ruházzuk fel. Mint a 2. ábrán is látható, a programrétegek között nincs közvetlen függőség, ha az adatcserék aszinkron módon folynak, tehát az alkalmazás egyes részeinek cseréje a teljes rendszer megzavarása nélkül is végrehajtható.

Összegzés

Egy eseményalapú alrendszer alapján újfajta programozási modellt állíthatunk fel, amelynek a segítségével a fejlesztők egyedülálló, nagy teljesítményű környezetet teremthetnek a folyamatok aszinkron végrehajtásához. Természetesen a soros programozási stílustól gyökeresen eltérő szemléletet kell elsajátítani, ám a programfejlesztés szempontjából sokkal jobban áttekinthető keretrendszer bőségesen kárpótol mindezért. A bonyolult programösszetevők egybeépítése és együttműködése is egyszerűbbé válik.

Egy ilyen alrendszer legfontosabb erénye az, hogy képes az aszinkron és a szinkron programkód egyesítésére ugyanabban az alkalmazásban – sőt a kétféle típus akár ugyanazon az eljáráson belül is elegyíthető. A kevert modellt alkalmazva – a tényleges környezettől függően – mindkét szemlélet előnyei kihasználhatók. Az újfajta modellt elsősorban biztonságos alkalmazások fejlesztésekor, illetve küldetéskritikus alkalmazások hosszú távú támogatásakor érdemes használni. Következő írásomban bemutatom, hogy az AEM-nek a Linux-rendszermagban történt megvalósításával hogyan vált elérhetővé az új modell támogatása, illetve mi módon használhatjuk fel alkalmazások fejlesztésekor.

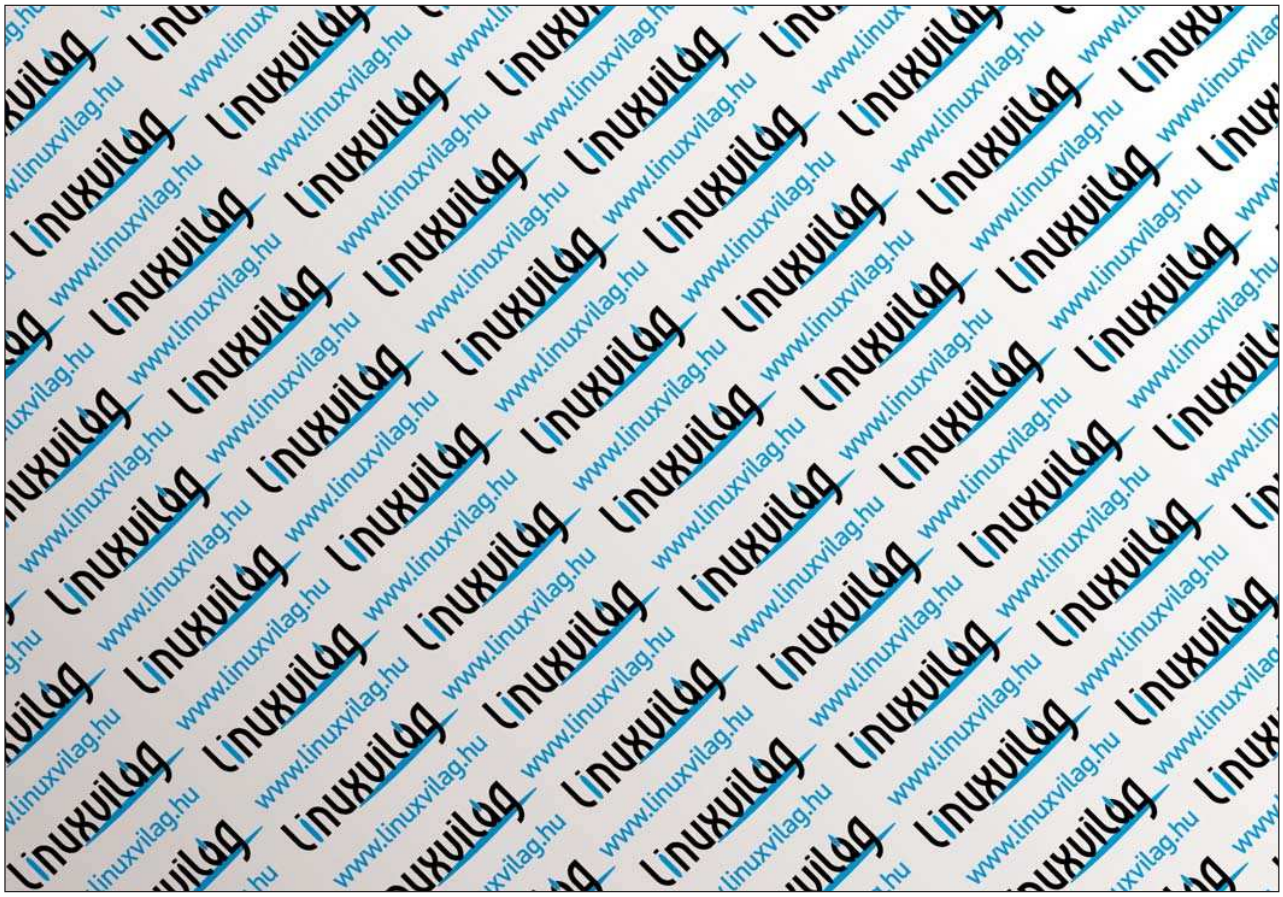
Köszönetnyilvánítás

Köszönettel tartozom az Open Systems Lab munkatársainak, amiért átnézték az írásomat és segítették a megjelenését; ***Laurent Marchand***-nak az Ericsson Research Canadától hasznos megjegyzéseiért, illetve ***Philippe Meloche***-nak, a Sherbrooke University hallgatójának.

Linux Journal 2003. július, 111. szám

Frederic Rossi

(frederic.rossi@ericsson.ca) Az Ericsson Research Open Systems Lab laboratóriumának kutatója a kanadai Montrealban. Részt vesz azokban a kutatásokban, amelyeknek eredményeképpen olyan rendszermag-összetevőket készítenek, amelyek lehetővé teszik egy távközlési szintű operációs kifejlesztését.



Hitelesítés LDAP használatával (2. rész)

A címtárkiszolgáló működik, tehát határozzuk meg a titkosítási beállításokat, majd adjuk hozzá a rendszerhez a felhasználókat.

A múlt hónapban már elvégeztük az OpenLDAP-kiszolgáló üzembe helyezésével járó munka egy részét. Telepítettük az alapot, vagyis a kiszolgálót, illetve a szükséges OpenLDAP-ügyfélcsomagokat, majd megadtunk bizonyos alapvető beállításokat a */etc/openldap/slapd.conf* fájlban (a `slapd` az OpenLDAP kiszolgálódémona).
Ez alkalommal a TLS alapú titkosítást állítjuk be, elindítjuk a démont, és megkezdjük egy LDAP-adatbázis felépítését. Teljesen ugyan nem fogjuk befejezni a hitelesítő kiszolgálót, ám nagyon közel kerülünk ehhez az állapothoz. A jövő hónapban megjelenő cikkben – ami egyben a három részből álló sorozat utolsó tagja lesz – a maradék munkát is letudjuk.

LDAP-tranzakciók TLS alapon

Az OpenLDAP adatcseréi alapesetben nyílt szövegek használatával folynak a hálózaton keresztül. Ha az OpenLDAP például központi telefonkönyv-kiszolgálóként teljesít szolgálatot egy megbízható hálózaton, akkor ezzel nincs is semmi gond. Ha azonban a hálózat megbízhatóságával nem foglalkozva felhasználók hitelesítésére használjuk, akkor jobb, ha a hallgatózók ellen titkosítással védjük az LDAP párbeszédeket, illetve egyben a felhasználók jelszavait.
Az LDAP v.3 protokoll, amelynek támogatása az OpenLDAP 2.0-s változatában jelent meg, képes a Transport Layer Security (átvitelirétegbeli biztonság, TLS) alapú védelemre – ezt a megoldást a böngészők és a levelezőprogramok is használják. A TLS az SSL (Secure Sockets Layer) utódja. Ha élvezni akarjuk a TLS nyújtotta előnyöket, az LDAP-kiszolgálón mindössze létre kell hoznunk egy kiszolgálótanúsítványt, hozzá kell adnunk néhány sort a */etc/openldap/slapd.conf* fájlhoz, majd egy picit meg kell piszkálnunk a `slapd` indítási beállításait.
A kiszolgálótanúsítvány létrehozásához OpenSSL-re van szükség. Ez valószínűleg már megtalálható a rendszereden, a bináris OpenLDAP-csomagok ugyanis az OpenSSL-től függenek.
Az, hogy LDAP-tanúsítványként pontosan milyen tanúsítványt érdemes használni, bizony, fogas kérdés. Olyan tanúsítványra van szüksége a kiszolgálónak, amelyet valamelyik hitelesítő szervezet (certificate authority, CA, mint például a VeriSign vagy a magyar NetLock) írt alá? Vagy a másik oldalról megközelítve: az LDAP-ügyfeleknek külső fél által ellenőrizhető tanúsítványt kell látniuk, amikor csatlakoznak a kiszolgálóhoz? Esetleg a saját szervezetünk önmaga hitelesítő szervezete lesz? Az utóbbi esetben helyi CA-ként is szolgál majd az LDAP-kiszolgáló, ami gondoskodik a saját tanúsítványának, valamint a többi állomás és felhasználó tanúsítványainak kibocsátásáról és aláírásáról?
Ha valamelyik kérdésre igen a válasz, akkor egy kicsit több dolgot kell elintézni, mint amennyit itt leírhatnék. Legyen elég annyi, hogy a `slapd` által használt tanúsítványhoz nem tartozhat jelszó – vagyis a kulcsa nem lehet DES kódolású –, így önaláírt tanúsítvány, hiába CA-tanúsítvány, valójában nem használható CA-tanúsítványként más tanúsítványok aláírására.
Ha azt akarjuk, hogy az LDAP-kiszolgáló valódi CA-ként működjön, akkor két kulcsot kell létrehoznunk: egy jelszóval védett CA-kulcsot és egy jelszó nélkül használható `slapd`-kulcsot. A témával ***Vincent Danen*** Using OpenLDAP for Authentication című cikkében foglalkozik bővebben (➲ http://www.mandrakesecure.net/en/docs/ldap-auth.php).
Az esetek túlnyomó részében elég egy saját TLS-tanúsítványt létrehozni, amelyet kizárólag a `slapd` használ. Ha CA-t nem akarunk létrehozni, illetve az LDAP-ügyfelek sem akarják külső fél segítségével ellenőrizni a tanúsítvány hitelességét, akkor a tanúsítványt az alábbiak szerint hozhatjuk létre:

```
bash-$> openssl req -new -x509 -nodes -out
➥slapdtanusitvany.pem -keyout slapdkulcs.pem
➥-days 365
Using configuration from
➥/usr/share/ssl/openssl.cnf
Generating a 1024 bit RSA private key
....++++++
.........++++++
writing new private key to ´slapdkulcs.pem´
```

A fenti paranccsal arra utasítottam az OpenSSL-t, hogy hozzon létre egy új, jelszavas védelem nélküli X.509 tanúsítványt, majd ezt (a nyilvános kulcsot) írja ki a pillanatnyi munkakönyvtárba, a ***slapdtanusitvany.pem*** nevű fájlba, a titkos kulcsot pedig a ***slapdkulcs.pem*** nevű állományba mentse. Élettartamként 365 napot adtam meg.
A parancs kiadása után meg kell adnunk a tanúsítványhoz és a kulcshoz tartozó megkülönböztető neveket. Az OpenLDAP esetében a legfontosabb mező a közös név (CN, common name). Ennek egyeznie kell az LDAP-kiszolgáló DNS-nevével. A tanúsítványhoz hozzárendelt névként az LDAP-ügyfelek ezt a nevet fogják látni. Ha például az LDAP-kiszolgáló IP-címéhez a ***bonzo.lamemoviesfromthepast.com*** név tartozik, ám a kiszolgáló tanúsítványban megadott CN értéke ***bonzo.lm.com***, akkor az LDAP-ügyfelek vissza fogják utasítani a tanúsítványt, és így képtelenek lesznek TLS-kapcsolatokat teremteni (az eredményt megjósolni nem lehet, az az adott ügyfélprogramtól függ).
Ha megvan a tanúsítvány és a kulcs, a fájlokat másoljuk a */etc/openldap* könyvtárba, feltéve, hogy eddig nem ezt használtuk munkakönyvtárként. Mindkét fájl tulajdonosa az `ldap` legyen, illetve az a felhasználó, akinek a neve alatt a `slapd` fut. Red Hat és SuSE alatt ez az `ldap`. A kulcsfájlra a legszigorúbb engedélyeket kell kiadni, például:

```
-r--------
```

A tanúsítványt tároló fájlt tulajdonképpen bárki olvashatja, hiszen nyilvános kulcs található benne.
Arra is van lehetőség, hogy a `-out` és a `-keyout` kapcsoló után ugyanazt a fájlnevet adjuk meg, ekkor a tanúsítvány és a kulcs ugyanabba az állományba kerül. Ennek akkor van értelme, ha a tanúsítványt nem akarjuk megosztani. A két különálló fájl használata ugyanakkor lehetővé teszi a kiszolgáló

A /etc/openldap/slapd.conf fájl testreszabott része

```
database        ldbm
suffix          "dc=proba,dc=org"
rootdn
"cn=ldapproba,dc=proba,dc=org"
rootpw
{SSHA}zRsCkoVvVDXObE3ewn19/Imf3yDoH9
directory       /var/lib/ldap
TLSCipherSuite  HIGH:MEDIUM:+SSLv2
TLSCertificateFile
/etc/openldap/slaptanusitvany.pem
TLSCertificateKeyFile
/etc/openldap/slapdkulcs.pem
```

tanúsítványának terjesztését, miközben a titkos kulcsot valóban titokként kezelhetjük.
Természetesen nem elég a tanúsítvány- és a kulcsfájl bemásolása a megadott könyvtárba, a `slapd`-t is utasítanunk kell a használatukra. A `slapd` beállításainak túlnyomó részéhez hasonlóan erről is a */etc/openldap/slapd.conf* fájlban rendelkezhetünk. A *listát* a múlt hónapban már megismert, azóta tán már el is felejtett ***slapd.conf*** fájl bejegyzéseit tartalmazza – ezekhez három új sor társul: `TLSCipherSuite`, `TLSCertificateFile` és `TLSCertificateKeyFile`.
A `TLSCipherSuite` azokat az OpenSSL titkosítási eljárásokat sorolja fel – a leginkább kívánatossal kezdve –, amelyek közül a `slapd` a TLS-kapcsolatok egyeztetésekor választhat. Azt, hogy a helyi gépre telepített OpenSSL-példány mely titkosítási eljárásokat támogatja, a következő paranccsal állapíthatjuk meg:

```
openssl ciphers -v ALL
```

A titkosítási eljárások felsorolása mellett az OpenSSL által támogatott helyettesítő szavakat is használhatjuk, így egy-egy szóval több eljárást is kiválaszthatunk. Az 1. kódrészletben például a `TLSCipherSuite` értéke `HIGH:MEDIUM:+SSLv2`. A `HIGH`, a `MEDIUM` és a `+SSLv2` kivétel nélkül helyettesítő szavak.
A `HIGH` jelentése: „minden olyan titkosítási eljárás, amelyik 128 bitesnél nagyobb kulcshosszal dolgozik”; a `MEDIUM` a „minden 128 bites kulcsot használó eljárás” rövidítéseként fogható fel, a `+SSLv2` pedig a „minden az SSL-protokollban megadott eljárás, tekintet nélkül a kulcshosszra” szinonimájaként kezelendő. Az OpenSSL titkosítási eljárások részletesebb ismertetését, illetve a támogatott helyettesítő szavakat a ciphers(1) súgóoldalon lehet megtalálni.
A `TLSCertificateFile` és a `TLSCertificateKeyFile` kapcsoló jelentése kézenfekvő: a tanúsítványt és a titkos kulcsot tartalmazó fájl nevét adják meg. Ha a tanúsítványt és a kulcsot ugyanabba a fájlba helyeztük el, akkor mindkét kapcsolónak ugyanazt az értéket kell adni.

A slapd indítási kapcsolói

A kiszolgáló oldalán minden szükséges lépést elvégeztünk annak érdekében, hogy a TLS titkosítás működjön. Most már csak egy kérdésben kell dönteni. A TLS használata az összes LDAP-kérés esetében kötelező legyen, vagy mint választható lehetőséget kínáljuk fel?
Alapesetben a `slapd` a 389-es TCP-kapun fogadja az LDAP-kéréseket, legyenek azok akár nyílt szövegben érkezők, akár titkosítottak. Ha az LDAP-t hitelesítési célokra akarjuk használni, valószínűleg érdemes kötelezővé tenni a TLS használatát. Ebben az esetben jobb, ha a `slapd` a nyílt szövegként érkező kéréseket csak a helyi hurokfelület 389-es TCP-kapuján keresztül fogadja, a TLS alapúakat pedig az összes helyi cím 636-os TCP-kapuján várja – ez egyébként az `ldaps` szabványos kapuja. Mindezt a `slapd -h` indítási kapcsolójával szabályozhatjuk, amely a `slapd` által a kérések fogadására használt URL megadására szolgál. A `slapd -h ldap://127.0.0.1/ ldaps:///` parancs hatására például a `slapd` a helyi hurokfelületen (127.0.0.1) az alapértelmezett kapun (TCP 389) át fogadja az `ldap`-kapcsolatokat, a többi helyi címen pedig az alapértelmezett `ldaps` kapun (TCP 636) keresztül várja az `ldaps` kéréseket.
Ha Red Hat 7.3 vagy újabb rendszert használunk, ez egyben az alapértelmezett beállítás is. A */etc/init.d/ldap* a */etc/openldap/slapd.conf* fájlban keresi a TLS-beállításokat, és ha talál ilyeneket, akkor a `-h` kapcsolót pontosan az iménti példa szerint alkalmazza. Ha SuSE 8.1 vagy újabb terjesztést futtatunk, akkor ezt a viselkedést a */etc/sysconfig/openldap* fájl módosításával érhetjük el. Az `OPENLDAP_START_LDAPS` beállításnak ***yes***, a */etc/init.d/openldap* fájlban található `SLAPD_URLS` beállításnak pedig ***ldap://127.0.0.1*** értéket kell adnunk. A változó megadása a parancsfájl elején található, alapértelmezett értéke a ***ldap:///***.
Más Linux-terjesztéseknél ettől eltérő módon történhet az indítási kapcsolók, köztük a `-h` átadása a `slapd` számára, de a fentiek alapján remélhetőleg nem lesz túl nehéz a kívánt kapuk beállítása.

Próba

Nos, valóban működik TLS alapú LDAP-kiszolgálónk? Egy gyors helyi próbával megkapjuk a választ. Először is indítsuk el az LDAP-t:

```
bash-$ /etc/init.d/ldap start
```

Ezután az `ldapsearch` parancs segítségével végezzünk el egy egyszerű lekérdezést a hurokfelületen keresztül:

```
bash-$ ldapsearch -x -H ldaps://localhost/
➥-b ´dc=proba,dc=org´ ´(objectclass=*)´
```

Természetesen a saját LDAP-kiszolgálónk neve nem `dc=proba,dc=org` lesz. Ha gondoljuk, ezt a parancsot egy távoli gépen is kiadhatjuk, ekkor a `-h` kapcsolónál a localhost helyett az LDAP-kiszolgáló nevét vagy IP-címét kell megadnunk. Ha az LDAP-kiszolgáló válaszul kiírja az – egyelőre üres – LDAP-adatbázis tartalmát, majd a ***0 Success*** felirat jelenik meg, akkor a próba sikeres volt.
Ha érvénytelen tanúsítványra utaló hibaüzenetet kapunk, akkor próbáljuk hozzáadni az alábbi sort az ügyfélgép */etc/openldap/ldap.conf* állományához:

```
TLS_REQCERT         allow
```

Ezzel engedélyezzük az OpenLDAP vagy az OpenLDAP alapú ügyfélprogram (például `gq`) számára, hogy önaláírt kiszolgáló tanúsítványokat is elfogadjon.

LDAP-séma

Pillanatok múlva megkezdhetjük az LDAP adatbázis feltöltését. A megfelelő eszközök segítségével – ilyen a `gq` és az `ldapbrowser` – jelentősen csökkenthető az LDAP-adatok bevitele és a felügyelete miatti álmatlan éjszakák száma. Csakhogy ahhoz, hogy ezeket az eszközöket használni tudjuk, először meg kell alkotnunk a megfelelő LDAP-sémát, és a történet itt kezd kacifántos lenni.

Esetünkben két LDAP-adattípussal érdemes foglalkozni, az egyik az attribútum avagy jellemző, a másik az objektumosztály. A jellemzők alkotják a rekordokat. Ilyen jellemző például a felhasználók levélcíme, beceneve, telefonszáma. Az LDAP-adatbázis tetszőleges számú jellemzőt kezelhet, akár saját jellemzőket is kitalálhatunk. Ahhoz azonban, hogy egy rekord egy adott jellemzőt tartalmazhasson, a rekordot a megfelelő objektumosztállyal össze kell rendelni.
Az objektumosztály a felépítendő rekord típusát írja le. Megadja, hogy az egyes rekordok esetében mely jellemzők megadása kötelező, és melyek hagyhatók el. Ennek alapján azt is gondolhatjuk, hogy könnyű dolgunk lesz, hiszen csak ki kell választanunk azt az objektumosztályt, amelyik az általunk fontosnak tartott jellemzőket tartalmazza, majd az összes felhasználórekordot ehhez az osztályhoz rendeljük hozzá. Az élet azonban sajnos nem ilyen egyszerű.
A gyakorlatban nagy valószínűséggel különféle objektumosztályok jellemzőit akarjuk majd használni. Semmi gond, vélhetjük, minden felhasználórekordhoz több objektumosztályt is hozzárendelünk, és tetszésünkre csemegézünk majd a jellemzők közül. Ebben is van valami, de a dolog nem tudható le ennyivel. A szükséges jellemzőket megadó objektumosztályok jó eséllyel különféle sémafájlokba vannak szétszórva (a sémafájlok szöveges állományok, ezek jellemzőket és rájuk hivatkozó objektumosztályokat tartalmaznak). Mielőtt tehát megkezdenénk saját, jó néhány objektumosztály-hivatkozást és még több jellemzőt tartalmazó rekordjaink létrehozását, először ellenőriznünk kell, hogy a */etc/openldap/slapd.conf* fájl az összes szükséges sémafájlra – ezek általában a */etc/openldap/schema* könyvtárban vannak – tartalmaz-e hivatkozást.
Tegyük fel például, hogy LDAP-kiszolgálókat azonosítási célokra akarjuk használni, ezért a `userid` és a `userPassword` jellemzőkről semmilyen körülmények közt nem szeretnénk lemondani. A `grep` segítségével hamar kideríthetjük, hogy a */etc/openldap/schema* könyvtár fájljai közül az `uid` az *inetorgperson.schema* fájl *MAY* listájában (a megengedett jellemzők közt), az `inetOrgPerson` objektumosztály alatt található. Ennek két vonzata van. Az első az, hogy a */etc/openldap/slapd.conf* fájlnak tartalmaznia kell az alábbi sort:

```
include /etc/openldap/schema/inetorgperson.schema
```

A második az, hogy amikor felhasználórekordot hozunk létre, akkor meg kell vizsgálni, hogy a `objectclass: inetOrgPerson` létezik-e.

Felhasználórekordok létrehozása és hozzáadása

A múlt hónapban már említettem a `gq`-t, amely számos terjesztésben megtalálható. Ugyancsak kiváló program az `ldapbrowser`, amely a ➲ http://www.iit.edu/~gawojar/ldap címen érhető el. Kezdésként azonban előfordulhat, hogy például a szervezetünk bejegyzését kézzel hozzuk létre. Ehhez készítenünk kell egy *ldif* fájlt, majd az `ldapadd` paranccsal be kell emelnünk a tartalmát az adatbázisba. Az *ldif* fájl egy olyan szöveges állomány, amely jellemző/objektumosztály megadásokat tartalmaz (soronként egyet), például:

```
dn: dc=proba,dc=org
objectclass: top
objectclass: organization
o: proba kiszolgalo
```

A fentiek szerint a *proba.org* szervezetet határozzuk meg. Megadjuk megkülönböztető nevét, összerendeljük a `top` (minden rekordnál kötelező) és az `organization` objektumosztállyal, majd megadjuk a szervezet nevét (proba kiszolgalo), ennél a két objektumosztálynál ez az egyetlen kötelező jellemző. A rekordot az alábbi paranccsal írhatjuk be az adatbázisba:

```
bash-$ ldapadd -x -H ldaps://localhost/
➥-D "cn=ldapproba,dc=proba,dc=org"
➥-W -f proba_adatok.ldif
```

Mint a legtöbb OpenLDAP parancsnál, a `-x` egyszerű jelszavas azonosítást ír elő, a `-H` az LDAP kiszolgáló URL-jét, a `-D` pedig a rendszergazdai fiók megkülönböztető nevét adja meg, míg a `-W` a rendszergazdai jelszó bekérését váltja ki. A `-f` kapcsolót az *ldif* fájl elérési útja követi.
Alakul? Lehet, hogy egy kicsit sok dolgot kellett most megemészteni, de vigasztaljon mindenkit az a tudat, hogy az LDAP-kiszolgáló kis híján készen áll.

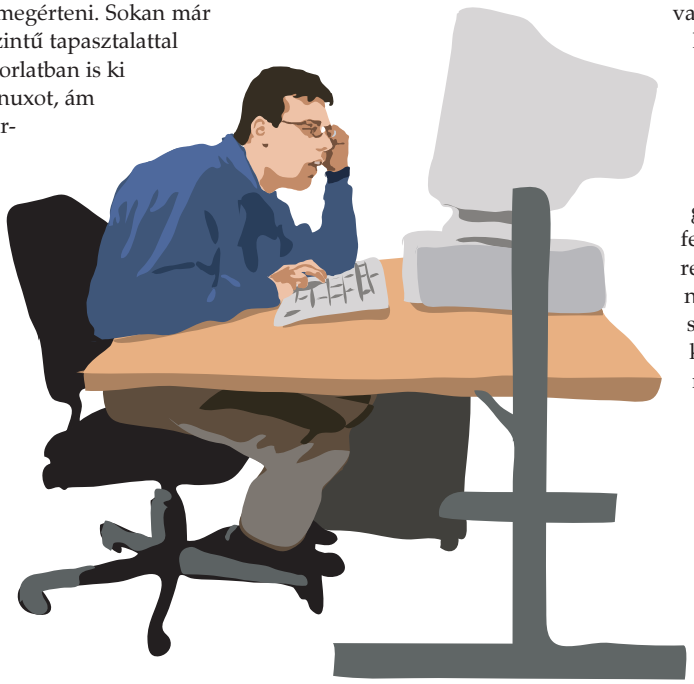
Linux Journal 2003. augusztus, 112. szám

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.

Bárki lehet rendszergazda!

Az SE Linux elkülönített védelmi rétege még a rendszergazdától is megvédi a rendszert. Russell betekintést nyújt a megoldás működésébe, és mindenkit rendszergazdává tesz.

Az NSA Security Enhanced Linuxon 2001 közepe óta dolgoztam, a debianos csomagok elkészítésében és az általános fejlesztésben egyaránt közreműködtem. Amikor Linux-felhasználóknak próbálom ecsetelni a tervezet lényegét, rengeteg félreértéssel szembesülök az SE Linux lényegét illetően. Az SE Linux valódi mibenlétét elég nehéz pusztán a leírásának átolvasásával vagy egy bemutató megtekintésével megérteni. Sokan már rendelkeznek valamilyen szintű tapasztalattal a biztonság terén, és a gyakorlatban is ki szeretnék próbálni az SE Linuxot, ám nincs idejük arra, hogy kísérletezés céljából feltelepítsék. Úgy véltem, a legjobb módszer az SE Linux népszerűsítésére az, ha üzembe helyezek egy bárki által szabadon használható gépet.



Az SE Linuxot normál összeállításon bemutatni nem túl izgalmas dolog, hiszen mindössze a `ps ax` és a `dmesg` az a két parancs, amelynél érzékelhető korlátozás van. Alapesetben a `ps ax` a jogosultság nélküli felhasználónak csak az azonos felhasználói tartományon belül futó folyamatokat mutatja meg, a `dmesg` pedig nem érhető el. Ezeket a korlátozásokat az `OpenWall` lépteti érvénybe, és önmagukban semmi újdonsággal nem szolgálnak. Úgy döntöttem, hogy a világ összes felhasználójának rendszergazdai hozzáférést adok, miközben a védelmet kizárólag az SE Linuxra terhelem, így képességeit a látogatók pontosan felmérhetik.

A 2002. június 6–9. között Karlsruhében, Németországban megrendezett LinuxTagon egy SE Linux bemutatógéppel jelentem meg a Debian pultjánál. Ez volt az első SE Linux „játékgép”. Akkoriban az alapértelmezett házirend még kevésbé volt szigorú. A `setuid` és a `DAC_OVERRIDE` használatát normál felhasználóknak is megengedte (`user_t` tartomány). Átlagos SE Linux-összeállításnál ezzel nincs is gond. Az SE Linux nem használ `uid` hívásokat a jogosultságok odaítélésekor, és ugyan a `DAC_OVERRIDE` lehetővé teszi a Unix hozzáférés-vezérlés felülbírálását, ám az SE vezérlésekét nem. Ez a két lehetőség azért volt elérhető, hogy a `setuidos` programok SE Linux-tartományok nélkül, az `user_t` tartományból is futtathatók legyenek. Az átlagos felhasználónak ez így meg is felelt, de nem akkor, amikor a rendszergazdától kellett megtagadni az azonos tartományban lévő más `uid`-ek elérését. A `user_t`-ből tehát kiszedtem ezeket a lehetőségeket, a rendszergazdai fiókot a `user_r` szerepre korlátoztam, és már kezdődhetett is a mulatság.

Az újabb kiadásokban az alapértelmezett házirend szerint a `setuid` vagy a `DAC_OVERRIDE` használatának lehetőségét a `user_t` nem kapja meg. Egykori játékgépem és egy élő kiszolgáló biztonsági házirendje között tehát az a legfontosabb különbség, hogy a játékgépen a jogosultság nélküli felhasználók elolvashatták a rendszermag üzeneteinek naplóját (`dmesg`) és a biztonsági házirend forrását, megkönnyítve ezzel az SE Linux megismerését.

A próbagépet szándékosan kevésbé biztonságosra állítottam be, mint ahogy egy valódi kiszolgáló esetében tettem volna: jogosultságot adtam a naplófájlok elérésére, a biztonsági házirend olvasására, valamint a jogosultság nélküli felhasználók is rendszergazdaként garázdálkodhattak. Talán ennek is köszönhető, hogy sikerült apróbb rést találni a védelem falán.

A LinuxTag első napján kihasználhatónak tűnő hiányosság jelentkezett a */boot* könyvtár állományaival. Az egyik felhasználó úgy gondolta, a LILO-térképfájlból megtudhatná a LILO jelszavát. Azonnal módosítottam a házirendet, és megtiltottam a */boot* könyvtár elérését, az ilyenfajta gondokat megakadályozandó. Természetesen, ha fizikailag sikerül hozzáférni egy géphez, akkor előbb-utóbb át lehet törni a védelmét, de ezt nyilván a lehető legjobban meg kell nehezíteni.

A rendezvény ideje alatt kezdtem el dolgozni a többféle felhasználói szerep támogatásán. Ennek eredeti oka az volt, hogy munkatársaim egyike komolyabb célokra is használta a játékgépet. Az összes fájlját elvesztette, mert azokat a `root:user_r:user_t` biztonsági környezetben, `uid root`-ként hozta létre, és a többiek is ezt használták a rendszer kipróbálására. Természetesen az `rm -rf /` paranccsal mindenki megpróbálkozott, és ezzel az ő állományai is az enyészeté lettek. Maga a rendszer nem sérült, ugyanis a */bin*, */etc* és egyéb

rendszerkönyvtárak nem választhatók le vagy írhatók a `user_t` által. Miután a barátom a `user1_t` tartományban kapott fiókot, a `user_t` tartomány rendszergazdájaként többé nem lehetett elérni a fájljait.
2002. június 17-én egy Cobalt Qube alapú SE játékgépet az interneten keresztül is – mindenki számára – elérhetővé tettem. Az első gép július 11-ig folyamatosan a hálózaton lógott. Való igaz, ez nem túl nagy eredmény, de a gépet folyamatosan figyelni kellett. Egy ilyen játékszer – ha megtörik, és én nem cselekszem azonnal – bármikor veszélyessé válhat bárki számára, beleértve engem, az internetszolgáltatómat és a használóit is. Vagyis amikor elutazom vagy nagyon leköt a munkám, akkor ki kell kapcsolnom.

A gép beállításai

A gép saját IP Tables-szabályokat kapott, megelőzendő a nemkívánatos, a helyi gépről kifelé irányuló hálózati forgalmat. Egy tűzfal mögé került, ami hasonló megszorításokat alkalmazott az adatátvitelre. Ezzel a módszerrel meg tudtam akadályozni, hogy bárki is belülről kóstolgathassa a tűzfalamat, amíg meg nem töri a játékgépet. Eleinte az SMTP-n kívül szinte minden kimenő kapcsolatot engedélyeztem, de hamarosan úgy döntöttem, csak egy webproxyra engedek adatokat kifelé. Másfajta hálózati hozzáférések céljára SSH-alagutakat lehetett használni. Megtiltottam az X továbbítását is, így ha valamelyik látogató véletlenül engedélyezte volna a saját gépén, a játékgép többi használója nem tudta volna támadni.

Mennyire volt biztonságos?

A gép alig egy napja volt fenn a hálózaton, amikor az egyik felhasználó jelezte, hogy a */etc/shadow* olvasható. A LinuxTag bemutatóján jeleztem, hogy ez a könyvtár szándékosan kívül maradt az érdeklődési körömön, de a hálózatra való csatlakoztatás előtt ki kellett javítani a hibát. A *shadow* fájl tehát `shadow_t` típusú lett, emiatt viszont módosítani kellett a `spasswd` burkoló programot és a hozzá tartozó SE házirendet. A `shadow_t` teljes körű támogatásának megvalósítása nem volt egyszerű, mivel sok esetben a programok a fájlokat újra létrehozva változtatják meg a */etc/passwd* és a */etc/shadow* fájlt, az alapértelmezett `etc_t` környezetbe helyezve őket. Jó megoldás lett volna, ha ezeket a programokat úgy módosítottam volna, hogy az `open_secure(2)` rendszerhívást használják a biztonsági környezetnek a fájl létrehozásakor való megadására. Ezt az ötletet azonban elvetettem, mivel a különféle biztonsági alkalmazásokkal rengeteget kellett volna dolgozni, megkockáztatva, hogy valamilyen hiba miatt megsérül a védelem. Ehelyett inkább írtam egy olyan burkolókódot, amely ezeknek a programoknak a futtatásában segédkezik, és a */etc/passwd* környezetét kilépésük után `etc_t`-re állítja vissza. Ezekre a programokra vonatkozóan a `shadow_t`-t választottam alapértelmezett típusnak arra az esetre, ha fájlt hoznának létre a */etc* alatt. A */etc/shadow* akár `etc_t` típust is kaphatott, a jogosulatlan rendszergazdai felhasználók nem tudtak beleírni. A `user_t` tartomány rendszergazdai felhasználói számára a fájl csak olvasható volt.
A következő napon valaki rájött, hogy a */dev/nvram* nem kapott megfelelő védelmet. Bárki tudta írni, így bármelyik felhasználó össze tudta volna zavarni a BIOS beállításait, lehetetlenné téve a gép elindítását. Akár az is előfordulhatott volna, hogy valaki olyan értékek átadására veszi rá a Qube BIOS-át a rendszermagnak, hogy a következő indításnál már legyengült védelemmel áll volna fel a rendszer. A Cobalt BIOS olyan műveleteket végez el, amelyeket más gépeken a rendszertöltő, például a LILO hajt végre. A házirend módosításával ezt a rést is azonnal betömtem. Fontos megjegyezni, hogy más géptípuson – jelentsen az másfajta processzort vagy kiépítést akár – hasonló apró módosítások végrehajtására lehet szükség a biztonsági házirendben, gondoskodva a */dev* könyvtárban lévő eszközcsomópontok védelméről.
A jelenlegi házirenddel elég kicsi a valószínűsége annak, hogy egy ilyen jellegű hiányosság gondot okozzon, mivel az eszközcsomópontokkal alapértelmezés szerint szinte semmilyen műveletet nem lehet végezni.
Néhányan aggódtak amiatt, hogy valóban jól állítottam-e be a jogosultságokat, és további megerősítést kértek afelől, hogy semmilyen jogszabályba ütköző dolgot nem művelnek, ezért a */etc/motd* tartalmát is módosítottam, biztosítva a látogatókat, hogy a gépet kifejezetten biztonsági próbák céljából csatlakoztattam a hálózatra. Kijelentettem azt is, hogy a védelmi rendszer bármilyen jellegű megtörése megengedett, akár a gép használhatatlanná válásának árán is, amennyiben az alkalmazott módszerről engem tájékoztatnak. Szintén kijelentettem, hogy a gépet nem szabad más gépek ellen irányuló támadások indítására használni, bár ezt a tűzfalszabályok segítségével is igyekeztem megakadályozni. Végül mindenkit megkértem, hogy szolgáltatásmegtagadási (DoS) támadást ne indítson a gép ellen, mert egyrészt semmi érdekes nincs benne, másrészt nem ezek kezelése a kísérlet célja.
Június 20-a után a játékgép működése meglehetősen eseménytelenül folyt. 2003 februárjában az OSDEM rendezvényen ismét megjelentem játékgépemmel a Debian pultjánál, és „szerezd meg a zászlót” versenyt hirdettem. Az érdeklődés elképesztő volt, nemegyszer akár harmincan is figyelték, ahogy valaki a védelmi rendszeren való áttöréssel próbálkozik. Egyikük sikerrel is járt, sikerült elérnie egy fájlt egy megadott nem rendszergazdai fiók alól, miután rendszergazdaként jelentkezett be. Ezt az `EDITOR` környezeti változó értékének módosításával és a `crontab -e` parancs futtatásával érte el. A `crontab` több SE-jogosultsággal futtatta a szerkesztőt, mint az normál esetben történt volna, és így tágabb hozzáférést engedett neki. Igaz, hogy ilyen módszerrel normál kiszolgálón nem lehetne eredményt elérni, hiszen az ismeretlen emberek még SE Linux alatt sem kapnak rendszergazdai hozzáférést, de azért módosítottam a `crontab` házirendjét, megelőzve a további hasonló eseteket. Azt sem szabad elfeledni, hogy a `crontab` alapú támadás egyetlen felhasználói szerepre korlátozódott. A más tartományokban lévő fiókok – például az általam a játékgép üzemeltetésére használt is – érinthetetlenek maradtak.
Az egyetlen folyamatosan fennálló gond az erőforrás-használat volt. Sokan úgy gondolták, hogy a fájlrendszer telítésével vagy az egyéb erőforrások felemésztésével el tudnak érni valamilyen eredményt. Azt hiszem, a DoS-támadásokra vonatkozó kérésem nem volt elég egyértelmű.
Ugyancsak érdekes kérdés volt: hogyan győzhetném meg a felhasználókat arról, hogy valóban rendszergazdaként használják a gépet. A GCC fent volt a gépen, és sokan saját `ps`-változatot vagy egyéb segédprogramokat hoztak, szentül meggyőződve arról, hogy valójában nem rendszergazdák, és csak valami ócska tréfát játszottam velük, módosított segédprogramokkal. Egyikük külön assembly kódot hozott a `getuid()` rendszerhívás használatára, így nyomozva az után, hogy vajon módosítottam-e a *libc6*-ot. Természetesen ő is valóban rendszergazda volt, de azért érdemes eljátszani a gondolattal: vajon hogyan kell módosítani a *libc6*-ot, hogy egy ténylegesen nem rendszergazdaként belépett személy annak is érezze magát?

Minden olvasót arra bíztatok, hogy próbálja ki. Természetesen nem mindenkit volt ilyen nehéz meggyőzni. Volt egy fehérgallérosnak tűnő figura, aki olyan gépek után kutatott, amelyekre egy támadócsomagot (rootkit) tudott felrakni. Nálam is próbálkozott, de rá kellett jönnie, hogy az őt érdeklő könyvtárakat (*/bin*, */sbin* és */etc*) és a bennük lévő fájlokat nem tudja írni. Tőlem kért támogatást a cucc telepítéséhez, de sajnos nem tudtam segíteni neki.

Hogyan végezhetünk saját biztonsági próbát vagy telepíthetünk saját próbagépet?

Ha saját biztonsági próbagépet szeretnénk összeállítani, akkor elsőként megfelelő helyet kell neki keresnünk. Ezt kimondani könnyű, véghezvinni viszont nehéz, ugyanis egy ilyen gép egy csomó hálózati pásztázást és behatolási kísérletet vonz a hálózatra. A legtöbb internetszolgáltató tiltja az ilyesmit, és jó eséllyel le fogja kapcsolni a gépet a hálózatról.

Ha a hálózati csatlakozást lerendeztük, ki kell találnunk valami jó módszert a gép hálózatról való leválasztására arra az esetre, ha valami balul sülne el. A kapcsoló közvetlen elérése például nem tűnik rossznak. A tápellátás vagy a RESET kapcsoló internetes vezérlésének lehetővé tétele is jó megoldás. Ha a távvezérlés nem kivitelezhető, akkor a próbagépet telepítsük egy felügyelhető kapcsoló (switch) külön kapujára, illetve ha ez sem oldható meg, akkor egy Netfiltert futtató linuxos géppel kössük össze keresztcsatolt (laplink) kábellel. Így pillanatok alatt megszüntethetjük a gép hálózati kapcsolatát.

Következő lépésként megfelelő vasat kell szerezni. Egy iPAQ például nem feltétlenül jó választás, mivel programból is teljesen használhatatlanná lehet tenni. Egy átlagos asztali PC nagyjából jónak tűnik. A legrosszabb esetben ki kell benne cserélni az alaplapot, ami kibírható költséget és munkát jelent. Ha esetleg sikerül ingyen szerezni egy gépet, akkor a rendszer teljes halála sem jelent különösebb megrázkódtatást, legalábbis anyagilag. Napjainkban úgyis kiváló eszközök kerülnek a szeméttelepre (az USA-ban – a szerk.).

Ha a gép alapvető beállításainak megadásán túlestünk, akkor megfelelő csomagszűréssel meg kell akadályoznunk, hogy más gépeket támadhassanak róla. Ezeknek a szabályoknak a szigorúsága az internetszolgáltatóval kötött szerződés tartalmának a függvénye. Ha ilyen jellegű hálózathasználatot a szerződés nem engedélyez – mert például otthoni célokra való széles sávú hozzáféréssel bírunk –, akkor szigorú szabályok szükségesek. Ha a szerződés lehetővé teszi kiszolgálók futtatását, akkor több dolgot engedhetünk meg, akár weboldalaknak is helyt adhatunk. Minél nagyobb szabadságot engedünk, annál érdekesebb próbákat lehet végezni. A felhasználók részéről az egyik leggyakoribb panasz az, hogy nincs elég mozgásterük a különféle próbák elvégzéséhez. A következő játékgépemen én már teljes hálózati hozzáférést szeretnék nyújtani, így a felhasználók például levelezhetnek majd a gépen, weboldalakat helyezhetnek el rajta – illetve amit még kérnek.

Tűzfalat a próbagépen és a fizikailag azonos hálózaton lévő összes gépen fel kell húzni. A próbagépen a Netfiltert értelemszerűen a csomagok csendes, naplózás nélküli elvetésére vagy visszautasítására kell beállítani, hacsak a naplók nem tarthatnak számot valakinek a kifejezett érdeklődésére. Az útválasztót úgy kell beállítani, hogy az összes eldobott csomagot naplózza, így hamar értesülhetünk arról, ha a próbagépen valaki sikeresen megkerülte a csomagszűrőt vagy a rendszer védelmét. Ha az internetszolgáltató is tud a próbagép felállításának tervéről, akkor egy egyszerű tűzfal is megteszi. Akadályozzuk meg az SMTP-kapcsolatok létrehozását, a hamisított forrás-IP-címmel küldendő csomagok továbbítását és a webes levelezőrendszerek használatát, például a Hotmailét, ami egyben a webproxyk elérésének megtiltását és a helyi webproxy megfelelő beállítását is maga után vonja.

A helyi hálózaton a próbagépen és az útválasztón kívül más gép lehetőleg ne legyen, ez ugyanis a próbagépről könnyen támadható lenne. Ha több próbagépet is sikerül ugyanarra a fizikai hálózatra kötni, az kellemes mulatság, mivel egymást lehet róluk támadni. Ha csupán egyetlen próbagépre futja, akkor keresztcsatolt ethernetkábellel vagy PPP-t futtató nullmodemmel csatlakoztathatjuk az útválasztóhoz.

A gépek összekötése és a tűzfalak beállítása után kezdődik a munka neheze. Meg kell határozni, milyen hozzáférési kört engedélyezünk, illetve a megfelelő naplózásról is gondoskodni kell. Az SE Linux esetében csak a felhasználói fájlok rendszergazdai bejegyzését kell módosítani, biztosítva a felhasználóknak a rendszergazda szerepet: `{ user_r };`.

Egy másik lehetőség a rendszergazdai bejegyzés teljes eltávolítása az adatbázisból, mivel az alapértelmezett `user_u` személyazonosság csak a `user_r` szerepben engedélyezett, és a jelszóváltoztatások megakadályozásával külön védelem biztosítható. Ha egy jogosultságok nélküli fiók jelszavát akarjuk megváltoztatni, a személyazonosságnak meg kell egyeznie a felhasználónévvel.

A változások a házirend-adatbázis újrafordításával és a rendszermagba való betöltésével léptethetők érvénybe. Ezt követően a rendszergazda már nem rendelkezik jelentősebb hozzáféréssel a rendszerhez, ezért ügyeljünk arra, hogy előbb egy másik fióknak adjunk felügyeleti jogot.

A következő próbagépem üzembe helyezésekor szeretnék jogi jártassággal rendelkező segítőt szerezni, aki átnézi a használati feltételeket, és biztosít arról, hogy a benne foglaltak egyértelmű és jogilag megtámadhatatlan módon írják le a megengedett műveletek körét. A jelszót a használati feltételekkel együtt egy weblapra fogom kihelyezni, és rendszeresen meg fogom változtatni, hogy a látogatók a feltételek elolvasása nélkül ne juthassanak be. Túl sok olyan belépő volt, aki nyilvánvalóan nem olvasta el a feltételeket, különösen a helyi – fork-bombákkal vagy éppen a merevlemez betöltésével végrehajtott – DoS-támadásokra vonatkozó részt.

Ha ilyen játékgépen futtatja valaki az SE Linuxot, akkor tegye meg azt a szívességet, hogy értesít, így írhatok róla a weblapomon.

A játékgépet használók támogatását, illetve az SE Linuxszal kapcsolatos kérdések megválaszolását a ➲ #selinux IRC-csatornán végeztem, az ➲ irc.debian.org kiszolgálón. Ha valaki ilyen biztonsági próbagépet állít össze, akár az SE Linux, akár más rendszer felhasználásával, akkor csatlakozzon a csatornához, ahol kicserélhetjük a tapasztalatainkat.

Köszönetnyilvánítás

A Sun Cobalt részlege egy RaQ kiszolgáló ajándékozásával segítette munkámat. A LinuxTag után az összes SE Linux játékgép Cobalt gépen futott.

Linux Journal 2003. augusztus, 112. szám

Russell Coker

Tíz éve használja a Linuxot. Internetszolgáltatóknál végzett Unix-rendszergazdai munkája során szembesült azzal, hogy a Unix a biztonság területén szorul a legtöbb fejlesztésre.

Sablon alapú XML-értelmezés C++ alatt

A Xerces könyvtár és egy kis C++-kód segítségével könnyen kezelhető objektumokba gyűjthetjük az XML-fájlokból az értelmezni kívánt adatokat.

Az XML kulcsszó alapú adatleíró nyelv, amit arra terveztek, hogy adatainkat beszédes, egyedileg választott kulcsszavakkal rendszerezhessük. Az XML feladata, hogy elkülönítse az adatot és annak felhasználását, illetve gép- és kiépítésfüggetlen módon tegye lehetővé az adatok mozgatását különböző alkalmazások között. Az XML egy másik hasznos alkalmazási lehetősége, amikor a folyamatokat logikus és átlátható módon írjuk le, és az alkalmazás futásidőben is végrehajthatja őket.

XML értelmezése

Ahhoz, hogy az XML-állományt sikeresen elemezhesse, a fejlesztőnek előbb létre kell hoznia egy olyan fájlt, amelyet az értelmező feldolgozhat. Az értelmező az XML-fájl beolvasását és értelmezését végző osztott objektumok készlete.
Kétfajta értelmező létezik: érvényesítő és nem érvényesítő. Az érvényesítő értelmező végignézi az XML-állományt és megállapítja, hogy az jól formált-e, megfelel-e a megadott XML-sémának vagy dokumentum-típusmeghatározásnak (DTD). A nem érvényesítő értelmezők egyszerűen beolvassák a fájl, és figyelmen kívül hagyják az XML-sémában vagy DTD-ben megadott szerkezetet.
Az általánosan használt értelmezők két eltérő megközelítést alkalmaznak: az eseményvezérelt és a faalapú megoldást. Az eseményvezérelt értelmezőt SAX-nak nevezik (egyszerű API az XML-feldolgozáshoz, angolul simple API for XML processing). A faalapú értelmező az XML-fájl beolvasása és értelmezése közben a memóriában egy DOM (document object model) fát hoz létre.
A DOM-megközelítésnél nehezebb eligazodni, és nem teszi lehetővé az XML-elemek és a tartományokhoz tartozó objektumok közötti egyszerű összerendelést. A SAX-események alkalmazásával lehetővé teszi a fejlesztők számára, hogy az XML-fájl beolvasása és értelmezése közben hozzák létre saját tartományra jellemző objektumait. Ebben a cikkben egy, az XML-értelmezéshez SAX API felületet használó keretrendszert ismerhetünk meg.

XML-értelmezők a C++ nyelvhez

A két legáltalánosabban használt C++ alapú értelmező az Apache projekt nyílt forrású Xerces rendszere és az IBM alphaWorks projektje keretében létrejött XML4C. Az XML4C a Xercesen alapul.
Mindkét értelmező lényegében ugyanazt a forrás- és könyvtárfelületet nyújtja, így felcserélhetők egymással. Egyaránt támogatják a DOM és a SAX alapú XML-értelmezést.
Az írásunkban bemutatott megoldás a Xerces értelmezőt és a SAX-megközelítést használja. Az XML-értelmezéshez kapcsolódó Xerces-források vagy bináris állományok a Xerces-weblapról tölthetők le (lásd a ***Kapcsolódó címeket***).

XML-fájlok értelmezése SAX-szal

Ahhoz, hogy az XML-állományokat elkezdhessük a SAX API segítségével értelmezni, előbb meg kell értenünk a SAX C++ objektumkapcsolatokat. A SAX-hoz két alapvető felületet terveztek (*1–2. táblázat*).

1. táblázat

SAXParser
setDoValidation
setDoNamespace
setDoSchema
setValidationFullSchemaChecking
setDocumentHandler
setErrorHandler
parse

2. táblázat

HandlerBase
warning
error
fatalError
startElement
characters
ignorableWhitespace
endElement

Tüzetesen megvizsgálva a `HandlerBase` objektumot, alapvetően két fajta tagfüggvényt figyelhetünk meg: vannak hibakezelő és dokumentumfeldolgozó tagfüggvények. A hibakezelő tagfüggvények közé tartozik a `warning`, az `error` és a `fatalError`, az értelmező tagfüggvények pedig a `startElement`, `characters`, `ignorableWhitespace` és az `endElement` lesznek. Mint később látni fogjuk, ezeket a viselkedési módokat külön objektumokra választhatjuk szét.
A `SAXParser` osztály felügyeli az alaptulajdonságok beállítását és a futásidőben érvényre juttatott, kívánt viselkedést.
A következő példakód bemutatja az XML-fájlok értelmezésének alaplépéseit, SAX értelmezővel C++ nyelven:

```
// SAX értelmező új példányának létrehozása
SAXParser parser;

// kívánt viselkedés alaphelyzetbe állítása
parser.setDoValidation(true);
parser.setDoNamespaces(true);
parser.setDoSchema(true);
parser.setValidationSchemaFullChecking(true);

// kezelők felvétele a dokumentumhoz és
// hibakezeléshez
parser.setDocumentHandler(&docHandler);
parser.setErrorHandler(&errorHandler);

// Fájl értelmezése
parser.parse("MyXMLFile.xml");
```

Az értelmezés pillanatában az általunk létrehozott osztályok (a `docHandler` és az `errorHandler`) továbbítódnak az értelmezés során kiváltott eseményhez. Ezek az osztályok a `HandlerBase` `Xerces` alaposztályból származnak, és felülírják a megfelelő tagfüggvényt, hogy a saját feladatkategóriájukhoz tartozó eseményeket kezelni tudják.
Most, hogy bemutattuk az XML SAX alapú értelmezését,

nézzük meg, hogyan készül el az XML-keretrendszer, és miképpen használja ki az API-ban rejlő lehetőségeket.

Rendszabályosztályok

Andrei Alexandrescu meghatározása a Modern C++ Design című művében (lásd a ***Kapcsolódó címeket***) ismertetett és híressé vált rendszabályosztályokra (policy class) a következő: „osztály-csatolófelületeket vagy osztálysablon-felületeket adnak meg. A csatolófelület egy vagy az összes elemet tartalmazza a következők közül: belső típusmeghatározások, tagfüggvények és tagváltozók."

A rendszabályosztályok igazi hasznát az XML-keretrendszerben akkor látjuk majd, amikor sablonalapú C++-rendszerben hozzuk létre őket. A rendszabályok segítségével egészen finom felbontással paraméterezhetjük és állíthatjuk be a képességeket. Ebben a felállásban a rendszabályok a következő feladatokat támogatják: dokumentumkezelés, hibakezelés, tartománykiosztás (domain mapping) és értelmezés (parsing).

Ha ezeket az elemeket rendszabályokként hozzuk létre, sokkal tömörebb kódot tudunk készíteni, amit egy C++ nyelvben és a sablonhasználatban jártas programozó könnyebben karbantarthat.

A XML-értelmező keretrendszer elsődleges osztálya az `XMLSAXParser`. Ez a testreszabhatóra tervezett osztálysablon tagváltozóként tartalmazza az `XMLParserInterface`-t és a `SAXParser` objektumot. A dokumentum- és a hibakezelők rendszabályosztályai egyaránt a sablonértékekhez tartoznak. Miután a megfelelő kezelőket és más tulajdonságokat beállítottuk, végső soron minden értelmezés a `SAXParser` tagváltozóhoz kerül.

Saját kezelőinket a keretrendszerrel szinte magától értetődő módon tudjuk rendszabályosztályként beilleszteni. Az ilyen típusú felépítésnek nagy előnye, hogy egy vagy több rendszabály megváltoztatásával ugyanazt a keretrendszert használhatjuk a különféle értelmező API-kban és eltérő tartománykiosztás-objektumokhoz – ezt a gyakorlatot azonban nem mutatjuk be cikkünkben.

Saját kezelők létrehozásához vezessünk le újonnan készített saját osztályokat a `HandlerBase`-ből, és írjuk felül a minket érdeklő virtuális tagfüggvényeket. A következő két saját kezelőtípus az `XMLFactory` keretrendszerben készült (*3–4. táblázat*).

3. táblázat

XMLSAXHandler

startElement
character
ignorableWhitespace
endElement

4. táblázat

XMLSAXHandler

warning
error
fatalError

Az `XMLSAXHandler` vezérli a dokumentumok eseményfeldolgozását, a `XMLSAXErrorHandler` a különböző hibavisszahívásokat kezeli.

XML-tagok és tartományobjektumok összerendelése

XML-értelmező keretrendszerünk következő feladata az XML-tagok átalakítása tartományfüggő objektumokká, amelyet azután az alkalmazásban sablonok és laza rendszabályosztály-meghatározások segítségével fel tudunk használni.

Az `XMLDomainMap` sablon egyetlen, `XMLNode` nevű sablonértéket fogad el. Tartomány-hozzárendelő objektumunk felülete a következőképpen néz ki (*5. táblázat*).

5. táblázat

XMLDomainMap

create
add
updateAttribute

Az `XMLNode` a gyermekeket alfákban egyesítő faszerkezetben egyszerre tölti be a gyökér és a levelek szerepét. Az `XMLNode` csatolófelület a következőképpen néz ki (*6. táblázat*).

6. táblázat

XMLNode

operator==
operator!=
operator=
addChild
hasChildren
numChildren
value
name
getChildCount
getChild
getParent

Az egész módszer kulcsa az objektum nyilvános felületének megtervezése. Megfigyelhetünk néhány műveleti-felülírást is, ilyen például az egyenlőségjel (operator==), a nem egyenlő (operator!=) és az összerendelés műveleti jel (operator=). Ezeknek az az előnye, hogy az objektumot mostantól rengeteg szabványos sablonkönyvtárral (STL), tárolóval és algoritmussal használni tudjuk, kiaknázva a C++ nyelv különleges képességeit.

Osztályok összefűzése – az XML Façade

Eddig különálló osztályokra és az XML-értelmező keretrendszerünkhöz készített sablonok ismertetésére összpontosítottunk. A következő lépésben összekapcsoljuk a különálló felületeket, így azok a façade tervezési mintának megfelelően a külvilág számára egyetlen, összefüggő egységnek látszanak majd.

7. táblázat

XMLProcessor

parse
getParseEngine

A façade tervezés egyszerű és elegáns módja annak, hogy a külső ügyfélről az értelmezéshez használt belső rendszabály osztályokba vigyük át az értelmezőszolgáltatást. A tervezési mintákban (Design Patterns) a szerzők a következőképpen foglalták össze céljaikat: „Célunk az alrendszer csatolófelület-készletéhez egységes felületet készíteni. A façade olyan magasabb szintű felületet határoz meg, amelynek segítségével az alrendszer könnyebben használhatóvá válik." A létrehozott façade neve XMLProcessor. Ezt a következő felülettel határozták meg (*7. táblázat*).

Miután minden forrást beírtunk, a példaprogram futtatásához szükség lesz egy XML-fájlra, illetve egy próbaügyfélre.

Az XML-fájl értelmezése

A keretrendszer egyszerű működésének bemutatására egy egyszerű (nevet és számlaszámot tartalmazó) vásárlóadatlapot leíró XML-állományt készítettünk:

```
<?xml version="1.0" encoding="iso-8859-1"?>

<customer>
     <name>John Doe</name>
     <account-number>555123</account-number>
</customer>
```

Készítsük el ezt az állományt valamilyen szövegszerkesztővel, és mentsük *MyXMLFile.xml* néven.

Nyilvános felület – ügyfélalkalmazás elkészítése

Az ügyfélalkalmazáshoz szánt tömör csatolófelületet a keretrendszer segítségével fogjuk elkészíteni.
Az ügyfélkeretrendszer által használható elsődleges tagfüggvények a következő, egészen rövid példa C++-forráskóddal írható le:

```
// ---------------------------------------
//  Példaforráskód az XML doc értelmezéséhez
// ---------------------------------------
#include "XMLProcessor.hpp"
#include "XMLDomainMap.hpp"
#include "XMLSAXParser.hpp"
#include "XMLNode.hpp"
#include "XMLCommand.h"
#include "XMLSAXHandler.hpp"
#include "XMLSAXErrorHandler.hpp"

#include <iostream>
using namespace std;
using namespace XML;

// Először is tegyük félre a ronda dolgokat
typedef XMLSAXHandler<XMLDomainMap<XMLNode> >
➥DOCHANDLER;
typedef XMLSAXErrorHandler ERRORHANDLER;
typedef XMLSAXParser<DOCHANDLER, ERRORHANDLER>
➥PARSER;
typedef XMLProcessor<PARSER> XMLEngine;

// alap-tesztügyfél létrehozása
int main(void)
{
     // a fájlnevünket hordozó
     // karakterlánc-objektum létrehozása
     std::string xmlFile = "MyXMLFile.xml";

     // Az XMLFactory egy példányának
     // létrehozása
     XMLEngine parser(xmlFile);

     //  rvényesítés kikapcsolása
     parser.doValidation(false);

     // XML fájl értelmezése
     parser.parse();

     // Fa gyökerének lekérése
     XMLNode root = parser.getXMLRoot();

     // Objektumunk nevének kiírása
     cout << "Root = " << root.name() << endl;
     return 0;
}
```

A fa gyökerének megfelelő `XMLNode` objektumpéldányunk értelmezése után a gyökér `XMLNode` gyermekei is elérhetővé válnak.

Példaügyfél fordítása

Az utolsó lépés az ügyfél fordítása lesz. A fordítást egyszerűen parancssorban végezzük:

```
g++ -o testClient -I.
➥-I/path/to/xerces/include
➥-I/path/to/xerces/include/xerces
➥testClient.cpp -L/path/to/xerces/lib
➥-lxerces-c
```

A fenti sor lefordítja nekünk az ügyfélalkalmazást. Ezután a teszt futtatása következik. Ne feledjük el úgy beállítani a `LD_LIBRARY_PATH` környezeti változót, hogy a helyi Xerces-telepítésünk ***lib*** könyvtárára mutasson. Mivel az osztott programkönyvtárak ebből a könyvtárból származnak, a hibátlan működés érdekében az alkalmazásbetöltőnek futásidőben valamilyen módon el kell tudnia érni a szükséges szimbólumokat.

A `testClient` futtatásakor a következőknek kell megjelenniük:

```
$>testClient

Adding child name
Adding child account-number
Root = customer
```

Mostantól egy teljes mértékben működőképes, C++-sablonokon alapuló XML-értelmező keretrendszerrel rendelkezünk, amelynek segítségével az XML technológiát beépíthetjük új, vagy már meglévő programjainkba. A példakód megtalálható a 52. CD Magazin/XML könyvtárában, illetve letölthető az ➲ ftp.ssc.com/pub/lj/listings/issue110/6655.tgz címről.

Linux Journal 2003. június, 110. szám

John Dubchak
Vezető programfejlesztő, aki tanácsadóként dolgozik Saint Louis környékén. Az elmúlt 12 évben C++ nyelven fejlesztett, és el sem hiszi, hogy az első C++ kódsorai mennyire rosszak voltak.

KAPCSOLÓDÓ CÍMEK

Xerces ➲ http://xml.apache.org/xerces-c
Xerces 2.1.0 (a cikkben ezt a változatot használtuk)
➲ http://xml.apache.org/dist/xerces-c/stable/xerces-c-src2_1_0.tar.gz
Andrei Alexandrescu Modern C++ Design, Addison Wesley Kiadó, 2002. 7–8. oldal
Erich Gamma, *Richard Helm*, *Ralph Johnson* és *John Vlissides* Design Patterns: Elements of Reusable Object-Oriented Software, Addison Wesley Kiadó, 1994.

Alkalmazzunk XML-t! (2. rész)

Sorozatunk első részében megismertük az xtend programot, valamint XML nyelvű bemeneti fájljait. A mostani részben további elemeket láthatunk, bepillantunk az XML belső működésébe, emellett az XML MakeFile használatát is kipróbáljuk.

Mint korábban láthattuk, a tagok meghatározásánál egy-egy tulajdonság értékére a '!név' forma használatával hivatkozhatunk. Erre létezik egy másik lehetőség is, ekkor a '?név' alakot használjuk. Ebben az esetben az adott tulajdonságot csak akkor írja be a kimeneti tagba, ha annak az értéke nem üres. Ilyen módon, ha a meghatározásban üres értéket adunk meg neki, akkor a kimenetben csak akkor jelenik meg, ha meghíváskor valódi értéket kapott.
Mielőtt megismernénk néhány további különleges tagot, megemlíteném, hogy mindig is nagy hatást gyakorolt rám a Forth nyelvnek az a sajátsága, hogy az alapvető (mag) részeken kívül a többi részt a legtöbb esetben már magában a Forthban írják. Ezért én is arra törekedtem, hogy a programom ilyen módon kívülről, XML-tagok révén bővíthető legyen. Ezért kézenfekvő volt, hogy egy olyan tagot készítsek (a neve <_code>), ami a tartalmát végrehajtatja a Perllel (ebből fakadóan ezt a programot nem célszerű mindenki számára futtathatóvá tenni az interneten át). Így anélkül, hogy magába a programba írnánk bele, jelentős mértékben bővíthetjük a képességeit. A továbbiakban ismertetendő különleges tagok jelentős része már így is készült. Nem hallgatom el azért, hogy néhányszor mégis bele kellett írnom magába a programba, de ez csak a megvalósítás tökéletlenségét mutatja, az elvét nem.
Példaként nézzük a legegyszerűbb tagot! Ennek mindössze az a célja, hogy ha nem akarunk mást, legyen egy bennfoglaló tagunk, amit az XML minden fájl esetén megkövetel. Ez a _dummy megtalálható az ***_dummy.xml*** fájlban.

```
<_code>
$r.=xpchild($stack[0]);
</_code>
```

Rövid magyarázat is elkél hozzá: a $r globális változóban gyűjtöm a kimenetet. A különböző tagok egymásba ágyazását úgy dolgozza fel, hogy a pillanatnyi tagot egy veremtárba helyezi el, és utána tér át annak „gyermek" (bennfoglalt) tagjaira. A veremtárváltozó neve stack, és ennek a legfelső (nulladik) eleme az éppen időszerű tag – a fenti sor ennek a bennfoglalt részét bontja ki (xpchild).
Hasonlóképpen az is lehetséges, hogy egy tag tulajdonságát szintén programkód segítségével állítsuk elő; ekkor a tulajdonságot az @ jellel kell kezdeni, és a kódnak egy szöveget kell eredményül adnia, ez lesz a tulajdonság értéke (például sprintf függvény).
Ha az <_code> tagon belül olyan karaktereket használunk, amelyek megzavarhatják az XML-elemző működését, akkor a kódot tegyük az előző részben már említett CDATA határolók közé, azaz

```
<![CDATA[     kód      ]]>
```

Ejtsünk még szót néhány olyan elemről, amelyek a programozási nyelvekhez hasonlóvá teszik az XML-bemenetet. Ezekkel a kimenet előállítása közben ciklust lehet szervezni, illetve feltételes kimenetet lehet előállítani. Természetesen igazi értelmüket akkor nyerik el, ha a következő részben ismertetendő adatbázis-, illetve grafikai illesztéssel együtt alkalmazzuk őket. Mindenesetre álljanak itt, alkalmazásukra utaló környezetben:

```
<_if test='$a<3'>
  ...valami, igaz ág...
</_if>
<_else>
  .... másvalami, hamis ág...
</_else>
```

A <_else> elemnek közvetlenül az <_if> elem után kell következnie!

```
<_while test='$a<5'>
 ... kimenetet előállító rész ...
<_code>$a+=1; </_code>
</_while>
```

Hasonló módon működik az <_until> is. Ezeken kívül a switch/case szerkezet is létezik:

```
<_switch test='$a'>
<_case test='1'> ..... </_case>
<_case test='valami'> ....  </_case>
<_default> ....  </_default>
</_switch>
```

Itt kell megemlítenem az <_ins> tagot is – ez teszi lehetővé, hogy az olyan részeket, amiket nem akarunk külön előállítani, hanem már készen állnak egy külső állományban, bemásolhassuk a kimenetbe. Ilyen lehet például a stílusleírás vagy a JavaScript, esetleg más betétek, részletek. A fájlnevet a következő módon kell megadnunk:

```
<_ins file='valami.scr'/>
```

Még egy érdekes eleme van az XML-nek, amit a későbbiek folyamán több esetben is alkalmazni fogunk. Még kellemesebbé teszi az a képessége, hogy ennek révén nemcsak a HTML-elemek, hanem a PHP-betétek, kódrészletek szempontjából is teljesen átlátszó lesz a program működése; ezeket is alkalmazhatjuk a bemenetben, és változás nélkül átkerülnek a kimenetbe (amit ebben az esetben ***.php*** kiterjesztésűre célszerű készíteni, így a kiszolgálónk megfelelően fogja értelmezni). A nyelvnek ez a tulajdonsága az úgynevezett feldol-

gozási utasítás (Process Instruction, azaz PI), formája pedig a következőképpen fest:

```
<?nev ... kód ... ?>
```

A név helyére a megfelelő PI nevét kell beírni, a kód helyére pedig bármit, amit az adott nevű PI-t feldolgozó program fogadni tud. Mint látjuk, ez (talán nem véletlenül) tökéletesen megegyezik azzal a móddal, ahogyan a PHP-kódot a HTML-oldalakba általában be tudjuk szúrni. Ebben az esetben a PI neve ***php***, ezt az `xtend` program külön kezeli: változtatás nélkül átmásolja a kimenetre.
A többi PI esetében másként járunk el. A névvel ellentétben ugyanis nem feldolgozási utasításnak használom őket, hanem arra, hogy a kimenet egy részét (a PI-kben megadottakat) bizonyos csatornákba irányítsam át. A PI-k tartalma ebben az esetben Perl-kód, amelynek a kiértékelése során állítjuk be a `$k` változót, és ennek értéke adódik hozzá a PI nevével címzett csatornához; például a következő esetben:

```
<?jsa1 $k=sprintf("parent.kep.zoom(1.2);\n";  ?>
```

A `"parent.kep.zoom(1.2);"` szöveg (plusz újsor) adódik hozzá a `jsa1` nevű csatorna tartalmához. Felmerülhet a kérdés, hogy mire is jó mindez. Nos, gondoljunk bele például abba, hogy a HTML-oldalakba sok esetben olyan részek vannak beágyazva, amelyeknek a helye meghatározott, viszont meghatározásainkban több elemnek is szüksége lehet arra, hogy ezekbe adatokat írjon. Ilyen például a stíluslap, ami vagy külön fájlban, vagy pedig a HTML-oldal fejrészében kap helyet. Ha nem lennének csatornák, akkor a HTML-oldal törzsében lévő elemek nem tudnának a fejrészbe adatot juttatni. Így viszont a következőképpen játszódik le a folyamat: a program két menetben olvassa a bemenetet: az elsőben előállításra kerülnek a csatornák tartalmai, a másodikban pedig ezeket szúrja be a megfelelő helyekre. A beszúró utasítás formája így fest:

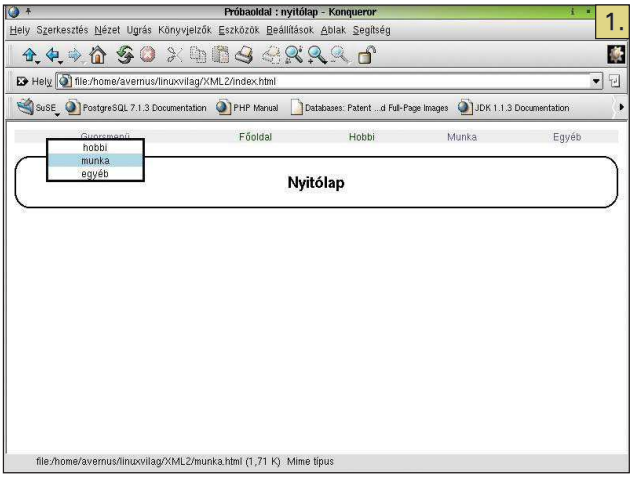
```
<_IR channel='jsa1' />
```

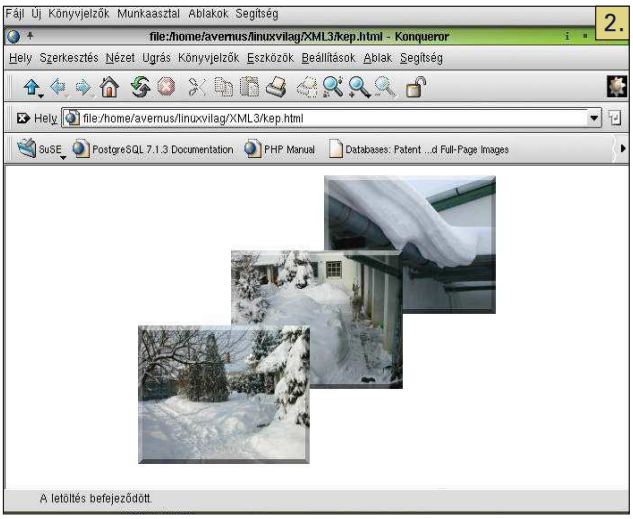
A `channel` tulajdonság értéke értelemszerűen a megfelelő csatorna neve. Még egy példa, ahol szükség van a csatornákra: a dinamikus HTML-oldalak előállításánál, ahol a JavaScript-betétek több (logikai) rétegben kerülnek elhelyezésre, és a különböző elemeknek bele kell írniuk egyikbe, másikba, többe is. Ez lineáris kimenetnél nem menne, a csatornákkal viszont egyszerű (példát a későbbiekben hozok rá).

A MakeFile.xml

A következőkben tekintsük meg a szintén xml formátumban írt `MakeFile` kialakítását. Mivel annyira magával ragadott az az elképzelés, hogy a program minden bemenő adata azonos formában kell legyen, és mivel amúgy is minden adott volt ehhez, a `MakeFile` formátuma ugyanaz lett, mint a többi bemeneté. Sőt gyakorlatilag ugyanaz a program elemzi ezt is, mégpedig néhány egyedi, e célból kialakított tag segítségével. Mi az előnye egy ilyen különleges fájl használatának azzal szemben, hogy a „valódi” `make` program „valódi” `Makefile`-jába írjuk be az előállítás feltételeit?

- Nem kell más formátumra figyelnünk (akinek nincs gyakorlata a `makefile`-ok írásában, annak ez elég komoly feladat).
- A kívánt honlap- vagy HTML-leírás oldalelrendezését is egy menetben meghatározhatjuk, ráadásul a későbbiekben ezt az adatot felhasználva a program különböző menüket és hivatkozásokat készíthet az oldalakhoz.
- Ugyanazt az egyetlen programot (`xtend`) kell meghívni az előállításkor.
- Változtatás esetén az oldal szerkezetének megfelelően újra létre lehet hozni akár egy oldalt is és az alárendelt (leszármazott) oldalakat.





Lássunk egy példát, ami némi magyarázattal kiegészítve megvilágítja a részleteket.

```
<site name='Próbalap' start='index.html'
            sitedir='./defs'>
<page name='nyitólap' src='proba.xml'
            target='index.html'>
<page name='hobbi' src='hobbi.xml'
            target='hobbi.html'/>
<page name='munka' src='munka.xml'
            target='munka.html'/>
<page name='egyéb' src='egyeb.xml'
            target='egyeb.html'/>
</page>
</site>
```

Mint látható, a `site` elem tartalmazza az összes többit. Itt adjuk meg a honlap nevét, az induló oldal fájlnevét, valamint

azt, hogy a második helyen keresett meghatározások hol találhatók (ebben az esetben a `MakeFile`-t tartalmazó könyvtárból nyíló ***defs*** alkönyvtárban).
Ezután jönnek az egyes oldalak meghatározásait leíró elemek, a `page` tagok. Figyeljük meg, hogy ezek az egymásba ágyazások révén faszerkezetszerűen írják le a teljes honlap szerkezetét. Példánkban a nyitólap a fa gyökere, és belőle nyílik a három másik oldal. Természetesen ennél sokkal mélyebben egymásba ágyazott, bonyolultabb szerkezet is előfordulhat.
Az egyes lapok esetében a három tulajdonság: az oldal neve (ezt bizonyos módszerekkel a böngésző címsorába is fel lehet vinni), a bemeneti fájl neve és a kimenet neve.
Ha ezek után a teljes honlapot elő akarjuk állítani a bemenetből, elég az alábbi parancsot kiadni:

```
xtend MakeFile.xml
```

Az `xtend` sorban előállítja a kimeneteket. Mivel eközben kiírja, hogy melyik bemeneti fájlból éppen melyik kimenetit készíti, nyomon követhetjük, hogyan járja be a fent említett faszerkezetet. Az előállítást a következő kapcsolókkal tudjuk irányítani:

`-global elérési_út` – másik keresési könyvtárat adhatunk meg.
`-site elérési_út` – a fenti `sitedir` tulajdonság helyett használható.
`-lsep 'karakter'` – a sortöréskarakter megadása, alapértelmezetten `\n`. Bizonyos esetekben, például ha olvashatatlanná akarjuk tenni a kimenetet, vagy bizonyos, erősen grafikus HTML-oldalaknál célszerű semmire ('') állítani át, így egyetlen sor lesz a kimenet, illetve a PI-k egy része ekkor is soremeléseket ír ki (ezt a JavaScript igényli).
`-x kimeneti_fájlnév` – csak a megadott és az abból leszármazott fájlokat hozza létre.
`-o kimeneti_fájlnév` – csak a megadott fájlt hozza létre (még a leszármazottakat sem).

Mivel a különböző oldalakat úgy állítja elő, hogy saját magát ismételten meghívja rájuk – azaz az `xtend` programot a megfelelő kapcsolókkal és nevekkel –, keresni kellett valamilyen megoldást arra nézvést, hogy a szükséges adatokat átjuttassuk a programhívások között (például a pillanatnyi oldal neve, a honlap neve). Ezek a hívás előtt a burok környezeti változóiban kerülnek elhelyezésre, és onnan tudja őket a program kivenni.
Ennyi szöveg után lássunk egy példát! A forrásszöveg a mellékletben található, itt csak a lényeges elemeket ragadom ki.
A nyitóoldal (***index.html***) forrása:

```
<alaplap>
<fejlec1/>
<br/>
<lap2>
<h2>Nyitólap</h2>
</lap2>
</alaplap>
```

Nagyon hasonlít hozzá a három másik oldal forrása is, csupán a `fejlec1` helyett `fejlec2` tag szerepel, és más a szöveg.
Az előállított eredmény az *1. képen* látható. A pillanatfelvétel időpontjában a gyorsmenü megnyitott állapotban található.
Ehhez a fenti kis forráson kívül másra is szükség van: egyrészről egy ***javacript*** könyvtárra, amit szintén e cikk szerzője készített, elemzése azonban meghaladja a cikk kereteit. A lényeg, hogy segítségével a 4.6-os Netscape, a 4.0-s Explorer és a 2.0-s Konquerornál frissebb, illetve az új DOM-mal (Document Object Model) egyező, JavaScriptet futtatni képes böngészőben ilyen menüket, valamint mozgó elemeket (sprite) lehet vele létrehozni. Emellett a fenti kis meghatározásban lévő egyéb elemeket is meg kell határoznunk. Példaként lássuk a `fejlec1` meghatározását (***fejlec1.xml***):

```
<_dummy>
<fwdmenu top='20' left='50' border-
color='#111111'/>
<fejfejhivatkozas>
<R><popa menu='fwdmenu'>Gyorsmenü</popa></R>
<R><a href='index.html'>Főoldal</a></R>
<R><a href='hobbi.html'>Hobbi</a></R>
<R><a href='munka.html>Munka</a></R>
<R><a href='egyeb.html'>Egyéb</a></R>
</fejhivatkozas>
</_dummy>
```

Az `fwdmenu` előre megadott elem, a `MakeFile` alapján készíti el a program, és a meghívó fájl összes leszármazott oldalát tartalmazza, ebben az esetben a fenti hármat. Ez nagymértékben a ***javascript*** könyvtárra épül. A `fejhivatkozas` egy olyan felsorolás, amelybe változó számú elemet lehet elhelyezni, és a kimeneten megismétli a meghatározásában megadott részeket. Ebben az esetben ezek táblacellák (td), teljes meghatározása a tábla leírását is tartalmazza.
Egy, az `fwdmenu`-höz hasonló feladatú elemet fedezhetünk fel a három beágyazott oldalon. Ez az úgynevezett `sitemenu`, tulajdonképpen az adott oldalhoz vezető útvonalat adja meg, hivatkozásként feltüntetve az elágazási pontokat. Ha megfelelő logikával készítettük el a `makeFile`-unkat, ezzel a kettővel megtakaríthatjuk a honlap „térképének” a különböző oldalakon történő elhelyezését, hiszen az `fwdmenu` megadja, hogy az adott oldalról hová mehetünk, a `sitemenu` pedig azt, hogy honnan jöttünk.
Hasonlóképpen a program a `MakeFile` alapján készíti el a böngésző címsorába kerülő címeket is. Ez, ha egyszer az `alaplap` elem meghatározásában a megfelelő kódrészletet elhelyeztük, önműködő.
A csatornák használatát az `alaplap` meghatározásában követhetjük nyomon. Ez írja ki a `jsb1`, `jsa1`, `jsa2`, `jsa3` és `style` csatornák tartalmát. Ezekbe a menüket előállító elemek írnak JavaScript-kódokat, valamint stílusadatokat.
Mint látjuk, egy viszonylag jól átgondolt elv szerint működő program esetén a már egyszer elkészített elemek újrafelhasználása nagymértékben leegyszerűsödik. Mégpedig nemcsak a ma szokásos kiszolgálóoldali programok módszerével, hanem statikus oldalak (például CD-re készülő katalógus) esetében is.
A tényleges megvalósításom hibákat és az elvtől való eltéréseket is tartalmaz. Ezekről és a program adatbázis-kezelő (PostgreSQL), valamint képfeldolgozó (ImageMagick) alkalmazásáról a következő részben olvashatunk.

Havránek Ferenc (hf@delvidek.hu)
Automatikamérnökként dolgozik. Kedvtelései közé tartozik mindenféle kétkerekű járművön (kerékpár és motor) való közlekedés. Ezenkívül szívesen tölti idejét programozással, nemcsak PC-s, hanem egyéb környezetben is, például mikrovezérlő programokat ír.

Írjunk többszálú programot Linux alatt! (1. rész)

A Qt könyvtár multithread-támogatásának bemutatása.

A Linuxon immár szabadon használható Qt fejlesztői keretrendszert bizonyára sokan ismerik a világon. Unix (Linux), az összes Windows, illetve Mac OS X operációs rendszerekre történő fejlesztésekhez egyaránt használható, de létezik olyan változata is, amelyik a beágyazott rendszerek fejlesztéséhez lett összeállítva. Fontos tudni azt is, hogy a KDE asztali környezetnek is a Qt csomag az alapja. A Qt-t sokan egy GUI-építéshez használható elemgyűjteményként, könyvtárként ismerik, ami olyan remek eszközökkel van felvértezve, mint a Qt Designer és a Qt Linguist. E tervezőeszközök tudása a Borland Kylixéhoz hasonlít. A Qt azonban sokkal több, mint egy grafikus felületet megvalósító program GUI-motorja.

Mik is azok a szálak?

A többszálú programozást lehetővé tevő környezetekben a szálak időosztásos vagy együttműködő módon futhatnak. Elsőként tisztázzuk, hogy mi is az a szál, illetve mit értünk többszálú program alatt. A hagyományos szekvenciális programok egyik tulajdonsága az, hogy mindig egyetlen aktív végrehajtási ponttal rendelkeznek, ami megmutatja, hogy mi lesz a következő végrehajtandó utasítás. Ezzel szemben egy többszálú program több aktív végrehajtási pontot tartalmazhat, ezek között az ütemező program osztja szét a CPU-erőforrást (általában több CPU is lehet). Ezek szerint a szál (thread) a folyamat (process) legkisebb önállóan ütemezett egysége. A szálak saját veremmel, gépi regiszterkészlettel, fontossággal (priority) és időszelettel rendelkezhetnek, de öröklik az őket befoglaló folyamat memóriacímterét, megnyitott fájljait, jeleit (signal) és általában a közösen használható erőforrásokat. A szálak használatának az a célja, hogy a folyamat párhuzamosan futtatható részei explicit módon is meg legyenek határozva, így a program-végrehajtási környezet rendelkezhet azzal az adattal, hogy mely tevékenységek párhuzamosíthatók. A szálak kezelése történhet mag- vagy felhasználói módban – mindkét felfogásnak megvannak a maga előnyei és hátrányai. A Linux több szálkezelő megoldással is bír, talán a legrégebbi a Posix `pthread` alrendszer. Ebben a cikkben a Qt 3.x által megvalósított, objektumközpontú szálkezelést ismertetjük, amit a Qt GUI-felületek dinamikusabb működtetése mellett az egyszerű textalapú programok készítésénél is használhatunk.

Szálak létrehozása és használata

A szálak létrehozásának megismerése közben írjunk egy olyan programot, ami kipróbálja, hogy a Qt könyvtár az időosztásos módszert alkalmazza-e a szálak ütemezésére. Az elv az, hogy elindítunk három szálat, ezek között semmilyen kapcsolat nincs, viszont mindegyik rendelkezni fog egy-egy helyi `szamlalo` változóval, ami folyamatosan növekszik, amikor éppen az őt tartalmazó szálnál van a vezérlés. A programot hagyjuk néhány másodpercig futni, majd valamilyen billentyű megnyomásával léphetünk ki belőle. A program utolsó lépéseként a `main()` függvény mindhárom szál `szamlalo` nevű helyi változójának értékét kiírja a képernyőre. Amennyiben a három szám hasonló nagyságrendű, úgy a Qt időosztásos ütemezést használ. Lássuk a programot (a CD-mellékleten ez a ***teszt0.cpp***)!

```
1. //
2. // teszt0.cpp
3. //
4. #include <iostream>
5. #include <qthread.h>
6.
7. using std::cout;
8.
9. // A TMyThread osztály felülete
10.class TMyThread : public QThread
11.{
12. bool leallitott; // A szál programozott
                      ➥leállítására
13. long int szamlalo; // helyi számláló
14. public:
15. virtual void run();
16. void leallit(void);
17. long int getSzamlalo() { return szamlalo; }
18.};
19.
20.// Programozott leállítás (Hasonló, mint a
Java 2 ajánlása)
21.void TMyThread::leallit(void)
22.{
23. leallitott = true;
24.}
25.
26.// Ez a szál kódja
27.void TMyThread::run()
28.{
29. leallitott = false;
30. szamlalo = 0;
31.
32. while ( !leallitott )
33. {
34. szamlalo++;
35. }
36.}
37.
38.
39.//--- A program indulási pontja ---
40.int main()
41.{
42. TMyThread t1, t2, t3;
43.
44. t1.start(); t2.start(); t3.start();
45. getchar();
46. t1.leallit(); t2.leallit(); t3.leallit();
47.
48. // A main() függvény bevárja, hogy az
       ➥összes szál lefusson
49. t1.wait(); t2.wait(); t3.wait();
```

```
50.
51. cout << "\nAz első szál számlálója: " <<
t1.getSzamlalo();
52. cout << "\nA második szál számlálója: "
            ➥<< t2.getSzamlalo();
53. cout << "\nA harmadik szál számlálója: "
            ➥<< t3.getSzamlalo() << "\n";
54.
55.
56.}
```

A program egyik futási eredménye a következő:

- Az első szál számlálója: 72 459 958.
- A második szál számlálója: 83 693 735.
- A harmadik szál számlálója: 81 453 880.

Látható, hogy a számlálók értéke azonos nagyságrendű, így a Qt önállóan kezelt időosztásos módban futtatja a szálakat. Nézzük végig a program forráskódját!
Az 5. sor mutatja, hogy a Qt szál (thread) használatához mindig a *qthread.h* beépített (include) fájlt kell beilleszteni. A 10. sorban egy `TMyThread` osztályt kezdünk létrehozni, ami nyilvános utódosztálya a `QThread` osztálynak. A `QThread` osztály valósítja meg a szál fogalmát a Qt könyvtárban. Osztályunknak két saját tagváltozója van. Az egyik a már ismertetett `szamlalo`. A másikat (melynek neve: `leallitott`) a programozott szálleállítási megoldás használja. Érdemes megjegyezni, hogy a Java 2 is ezt a fajta `iarendesenl`, programozott szálleállítási megoldást ajánlja. A 17. sor `getSzamlalo()` tagfüggvénye a `kiiráskor` lesz hasznos, hiszen a mi számláló adattagunk `privat`, így csak egy ilyet elérő függvénnyel kérhetjük le az értékét. A 20–24. sor a szálleállító eljárást valósítja meg. Ebben a kódban nem volt fontos ennek a változónak a kizárólagos használata. A 27. sortól kezdődik a `public` elérésűre meghatározott `run()` tagfüggvény, ami a végrehajtási szál kódját tartalmazza, és feladata lényegében csak a számláló folyamatos növeléséből áll. A 40. sorban kezdődő főprogram `t1`, `t2`, `t3` néven három szálat hoz létre, majd a 44. sorban a `QThread` osztálytól örökölt `start()` tagfüggvénnyel elindítja őket. Ezután a főprogram a `getchar()` hívásnál elakad, és arra vár, hogy valaki leüssön egy billentyűt. Eközben a `t1`, `t2`, `t3` szál dolgozik. Egy billentyű megnyomása után a 46. sorban leállítjuk a szálainkat. Miért van szükség a 49. sor `wait()` hívásaira? Mi az a `wait()` tagfüggvény? A `QThread` osztály `wait()` tagfüggvénye a szálak összekapcsolását (`JOIN`) valósítja meg. Legyen `sz1` és `sz2` két szálváltozó. Ekkor az `sz1` `run()` kódjából hívott `sz2.wait()`-nek az a hatása, hogy az `sz1` szál addig blokkolódik, amíg az `sz2` véget nem ér, mely után az `sz1` folytatja a munkáját. Az `sz1` szál tehát így tudja bevárni `sz2` befejeződését. A mi programunkban is ez a cél. A `main()` függvénynek mindegyik szál befejeződését be kell várnia, és ő maga csak ezután fejeződhet be, de előbb az 51–53. sorban kiírja a számlálók értékét. A Qt GUI-alkalmazásokban a `wait()` használata nem szükséges, hiszen úgyis csak egy kiléptető eseménnyel fejezhetjük be a programunkat.
Befejezésül nézzük meg, hogyan tudjuk lefordítani a *teszt0.cpp* programot. A hozzá tartozó `Makefile` az 52. CD Magazin/Qt könyvtárában található, a HOGYAN-fájl a honlapunkon olvasható.

A szálak ütemezésének hangolása

Láttuk, hogy a Qt-szálak önműködően időosztásosak, sokszor jó lenne azonban az időszeletek hosszát saját kezűleg is befolyásolni. A *teszt1.cpp* programot az előzőek alapján már valószínűleg mindenki érti. Létrehozunk két szálat a főprogramban, majd mindkettő folyamatosan kiírja a szálazonosítóját, illetve azt, hogy hol tart a 3000-ig történő számolásban.

```
1. //
2. // teszt1.cpp
3. //
4. #include <iostream>
5. #include <string>
6. #include <qthread.h>
7.
8. using std::cout;
9. using std::string;
10.
11.class TMyThread : public QThread
12.{
13. string szalnev;
14.public:
15. TMyThread(string szalnev) { this->szalnev
            ➥= szalnev; }
16. virtual void run();
17.};
18.
19.void TMyThread::run()
20.{
21. for(int i=0; i<3000; i++ )
22. {
23. cout << "\n " << szalnev << " = " << i;
24. //if ( i % 100 == 0 ) QThread::usleep(100);
25. }
26.}
27.
28.//
29.// Indulás...
30.//
31.int main()
32.{
33. TMyThread t1("T1 szál");
34. TMyThread t2("T2 szál");
35. t1.start(); t2.start();
36. t1.wait(); t2.wait();
37. cout << "\n";
38.}
```

A program lehetővé teszi a szálak vizsgálatát, azaz egy fájlba átirányítva az eredményt megtekinthető, hogyan alakult a `t1` és `t2` szál aktív futási ideje. Ennek finomítási lehetősége az, ha a 24. sorból kivesszük a megjegyzést, és úgy futtatjuk a `teszt1.exe` programot. Az `i` ciklusváltozó minden 100. értékénél az éppen aktív szál önként lemond a futásáról, ennek eredményeképpen a másik folytatni tudja a futását. A futási eredmény mutatja, hogy a szálak időbeni aktivitása finomodott (becsempésztünk egy kis együttműködő jelleget az ütemezésbe), fontos azonban kiemelni, hogy ezzel a lehetőséggel a végsőkig visszaélve a hatékonyságot is ronthatjuk, hiszen növekszik a szálak közötti kapcsolgatás felügyeleti költsége.
Sorozatunk következő részében a QThread osztály részletes áttekintésével folytatjuk, továbbá szót ejtünk a közösen használható erőforrások védelméről is.

Nyíri Imre (inyiri@mol.hu)

A Pixelview PlayTv Pro tévékártya telepítése

Bizonyára nem én vagyok az egyetlen, aki tévét is szeretne nézni a számítógépén. Ehhez mindössze egy TV hangolóval szerelt médiakártyára van szükség, ami manapság már sokféle összeállításban és árkategóriában kapható.

Még a vétel előtt érdemes utánanézni a kártya Linux-támogatottságának, és ennek fényében megvenni az eszközt. Én egy Pixelview PlayTv Pro kártyát vettem, most ennek telepítésével ismertetem meg olvasóinkat. A kártyát behelyezve elindítottam a gépemet. A Linux azonnal megorrolt a hangkártyámra, és valós IRQ-hibát írt ki. A hibának utánanézve rájöttem, hogy igaza van – mint kiderült, a Geforce2, az SB 128 PCI és az új tévékártya is az IRQ 11-et használta. Lévén a hangkártya volt a legalsó PCI-sínben, arra haragudott meg rendszer. Nem volt mit tenni, ki kellett venni; még jó, hogy senkinek nem kellett a régi SB 16 vibra PnP ISA-sínes hangkártyám, és még a fiókomban volt. Gyorsan megnéztem a leírását, és láttam, hogy IRQ 5-öt használ. Tehát erre cseréltem le SB 128-as kártyámat (ezt azért írom le, mert értékes tapasztalat, erre is oda kell figyelni vásárlás előtt!). Így már minden gond nélkül indult a hangkártya is, és nekiláthattam a telepítésnek. A SuSE 8.2 eszközfigyelője rögtön szólt, hogy tévékártyát talált, be szeretném-e állítani. Természetesen szerettem volna. A YAST2 indult el, ez volt a segítségemre ebben. Jó tudni, hogy a tévékártya beállításának alapfeltétele egy hangkártya megléte, ennek hiányában le kell mondanunk arról, hogy a tévét beállítsuk. Miután megadtuk neki a hangkártyát, egy eszközlistához jutunk, innen választjuk ki a kártya típusát. Az „egyéb gyártók”-ra kattintva a jobb oldalon megjelenő listában megtaláljuk a Pixelview PlayTv Pro kártyát. Ezután viszont el is akadtam, mert be kell állítani a hangolót (tuner), ami független a kártyalapka-beállításoktól. Tehát a tévékártya vételénél két dolgot kell tudni (az IRQ-használaton kívül): a lapka és a hangoló (tuner) típusát. Itt akadtam el – ugyanis a gyártó honlapján sem találtam erről adatot. Azt megtudtam, hogy a lapka bt878 alapú (fontos, hogy Linuxon csak a bt84x és a bt87x alapú kártyákat lehet működésre bírni), sőt a pontos típust is kiderítettem: PV-bt878P+, de a hangoló típusát sehol nem találtam meg. Végül ugyanezt megpróbáltam UHU Linuxban is, mivel mielőtt megvettem a kártyát, kinyomtattam az UHU Linux-füzet tévékártyákra vonatkozó részét, egészen pontosan az eszköz- és a hangolótámogatottsági listát.
Az utóbbiban teljesen „eltévedtem”, hiszen semmit nem tudtam a hangolóról. Marad a legegyszerűbb megoldás: a próbálkozás. A gond megoldására a `modprobe` parancs a legjobb. Mivel a Linux észreveszi, hogy bt8xx alapú kártya van a gépben, önműködően betölti a `bttv` modult. De ez nincs paraméterezve, ezért nem jó. Jelentkezzünk be tehát rendszergazdaként (a `su` parancs kiadása a konzolban), majd távolítsuk el a modult:

```
# rmmod bttv
```

Ezután tudjuk csak újra betölteni a modult a megfelelő értékekkel. Tudjuk (a listából), hogy a Pixelview PlayTv Pro száma 37, és nincs rajta rádióhangoló. Akkor a parancs így néz ki:

```
modprobe bttv card=37 radio=0
```

Ez nem bizonyult jó választásnak. Próbálkoztam még, hogy a PlayTv vagy a PlayTv Pak számaira mit is válaszol. Lévén semmilyen tapasztalatom nem volt még tévékártyákkal, igencsak meglepett, hogy a PlayTv Pro száma (37) csak elvben jó. Tudniillik, amint 50-et adtam meg neki, a Zapping a devices infónál rögtön kiírta a teljes lapka nevét (PV-BT878P+4E). Ennek fényében kiemelném, hogy nem a logikusnak tűnő PlayTv Pro (card=37), hanem a PlayTv Pak (card=50) a helyes érték.
Ezután már az adás is bejött, szinte azonnal megjelent a királyi televízió 1-es csatornája. Igaz, a későbbiekben erősen hangolni kellett, de ott volt a kép. Jó tapasztalat volt, jól tükrözi, hogy olykor a felmerülő gyanú bizonyul igaznak.
Ezután kell betölteni a hangolót. Ha ezt nem tudnánk, marad a lista alapján a próbálkozás.
A listában 25 hangoló szerepel (0–24). Elsőre sikerült a jó értéket betölteni, mivel az ehhez a kártyához tartozó hangoló száma a 0. (Figyelem, ha elrontunk valamit, a tiszta, átlátható folyamat és a megbízható eredmény érdekében a modult az `rmmod bttv` paranccsal mindig távolítsuk el, és a `modprobe bttv card=50 radio=0` paranccsal kezdjük újra a folyamatot.)
A hangolóbeállítás parancsa így fest:

```
modprobe bttv type=0
```

A táblázat alapján láthatjuk, hogy ebben az esetben a kártya a "Temic PAL (4002 FH5)" hangolótámogatást használ.
Érdemes megjegyezni, hogy a kívánt eredmény a `bttv` dokumentációban leírt egyik LG hangolómeghajtóval is elérhető. Működik tehát a `type=28` is.
Ha ez is megvan, a tévékártyához a hangot az alábbi parancscsal kell aktiválni:

```
modprobe msp3400 amsound=1 once=0
```

Ha mindent jól csináltunk, akkor a zappingot elindítva, a beállítások/devices infóban látjuk kártyánk pontos típusát

```
(bt878( PV-BT878P+4E).
```

Ha rosszul csináltunk valamit, akkor a `bt878(UNKNOWN/GENERIC)` sort látjuk itt. Ekkor nyilván elölről kezdjük a folyamatot, esetleg újra áttekintjük a hangoló vagy a kártya számát.
Ha jól csináltuk, akkor a Zapping egy nagy „hangyafocit” mutat, és erős sistergéssel adja tudtunkra, hogy működik, de nincs csatorna. Én megpróbálkoztam a szobaantennával, és kézzel (gyenge jelnél az önműködő csatornabeállító nem veszi észre a csatornát) beállítottam egy csatornát, ekkor egy alig hallható hang jelent meg, de a kép továbbra is rossz és nagyon sistergős volt.
Tipp: ha gondjaink lennének a hanggal, akkor érdemes átnézni a `bttv` dokumentációt, és bővíteni a `bttv` értékeit az `audioall=1` és az `autoload=1` kapcsolókkal. Ez esetben

1. táblázat **A Linux által támogatott kártyák listája és azok száma**

Kártya (card=X)	Név
0.	***UNKNOWN****
1.	MIRO PCTV
2.	Hauppague old
3.	Minden más egyéb
4.	Intel
5.	Diamond DTV2000
6.	AverMedia TVPhone
7.	MATRIX-Vision MV-Delta
8.	Fly-Video II
9.	TurboTV
10.	Hauppague new (bt878)
11.	MIRO PCTV pro
12.	ADS Technologies Channel Surfer TV
13.	AverMedia TVCapture 98
14.	Aimslab VHX
15.	Zoltrix TV-Max
16.	Pixelview PlayTv
17.	Leadtek WinView 601
18.	AVEC Intercapture
19.	LifeView FlyKit w/o Tuner
20.	CEI Raffles Card
21.	Lucky Star Image World ConferenceTV
22.	Phoebe Tv Master + FM
23.	Modular Technology MM205 PCTV, bt878
24.	Askey/Typhoon//Anubis Magic Tview CPH501/061 (bt878)
25.	Terratec/Vobis TV-Bootstar
26.	Neewer Hauppauge WinCam (bt878)
27.	MAXI TV Video PCI2
28.	Terratec TerraTV+
29.	Imagenation PXC200
30.	FlyVideo 98
31.	iProTV
32.	Intel Crrate and Share PCI
33.	Terratec TerraTValue
34.	Leadtek Winfast 2000
35.	Chronos Video Shuttle II
36.	Typhoon Tview TV/FM Tuner
37.	PixelView Paly TV pro
38.	Tview99 CPH063
39.	Pinnacle PCTV Studio/Rave
40.	STB2
41.	AverMedia TVPhone 98
42.	ProVideo PV951
43.	Little OnAir TV
44.	Sigma TVII-FM
45.	MATRIX-VisionMV-Delta 2
46.	Zoltrix Genie TV/FM
47.	Terratec TV/Radio+
48.	Dynalink Magic TView
49.	GV-BCTV3
50.	Prolink PV-BT878P+4E (Pixelview playtv pro pak)
51.	Eagle Wireless Capricorn2 8bt878A)
52.	Pinnacle PCTV Studio Pro
53.	Typhoon Tview RDS / FM Stereo
54.	Lietec 9415 TV
55.	BESTBUY Easy TV
56.	FlyVideo '98/FM
57.	GrandTec "Grand Video Capture"
58.	Phoebe TV Master Only (no FM)

így néz ki a `bttv` parancsa. És ekkor felesleges a `modprobe msp3400 amsound=1 once=0` parancs.

```
#modprobe bttv card=50 audioall=1
➥autoload=1 radio=0
```

További tipp – ami csak azért tipp, mert a vezérlőlapka valódi kiadási változatától függ – a `card=72`. Ez ugyanis csak és kizárólag a Prolink `PV-BT878P rev.9B` esetén működik. De ekkor pluszkapcsolóként meg kell adni az `AMSEL=0`-t is. Miután erről meggyőződtünk, a fent leírt parancsokat érdemes egy indító parancsállományban elhelyezni, így a kártyát nem kell minden indítás után kézzel beállítgatni, hanem a rendszer ezt már az induláskor megteszi.
Fontos, hogy az indító parancsállományból ne maradjon ki a `bttv` modul eltávolítása sem, és ez legyen az első. Minderre azért van szükség, mert önmagában a `bttv` betöltése (némelyik Linux ezt – amint a kártyát „meglátja” – önműködően megejti) még nem alkalmas a kártya használatba vételére, mivel nincs valódi értéke. Tehát a indító parancsállományban elhelyezett sorok így néznek ki nálam, egy bt878 alapú, PV-BT878P+ lapkás Pixelview PlayTv Pro tévékártya esetében (rádió nélkül):

```
rmmod bttv
modprobe bttv card=50 radio=0 (rádióhangoló
esetén itt 1-es az érték)
modprobe bttv tuner type=0 (vagy 28)
(hangolóbeállítás)
modprobe msp3400 amsound=1 once=0
(hangtámogatás aktiválása)
```

Ha ez is megvan, akkor az indító parancsállomány mentve van, és számíthatunk rá, hogy minden indítás után a kártya működni fog. Erről könnyen meggyőződhetünk, ha a gépet újraindítjuk, és a betöltődés után elindítunk egy tévéprogramot. Nekem nagyon tetszett a Zapping, mert ismeri a teletextet, tud állóképet készíteni és MPG1-be felvenni (ehhez legalább egy 700–800 MHz-es gép javasolt, mert nálam az 525 MHz-es Celeronon egy kicsit darabosak a felvett MPG-filmek). Jó képet ad, és nagyon sok beállítási lehetősége van. Érdemes megemlíteni, hogyha a kártyához veszünk távirányítót (plusz 1–2 ezer forintos költség), akkor a Zappingban ennek a támogatását is aktiválni lehet.
A ***plugin*** részben kell megszemlélni, ahol látni fogjuk, hogy az állókép és a felvétel szolgáltatás is használható bővítményként a programban. A távkapcsoló bővítménye a `lirc`, amit el kell indítani. Sőt a programban szerkeszteni tudjuk, hogy a távkapcsoló melyik gombja legyen aktív, illetve inaktív. Immáron semmi akadálya nincs a tévénézésnek, a tévéből való képlopásnak, esetleg a műsor felvételének. A beállításokkal részben azt is be tudjuk szabályozni, hogy milyen kódolóval milyen formátumban vegye fel az anyagot. Érdemes egy kicsit elügyködni vele, mert egy optimális és jól beállított értékkel erőforrást takaríthatunk meg, szintén elfogadható felvételi minőség mellett.
Sikeres beállítást és jó tévézést kívánok mindenkinek!
A cikkhez tartozó 2. táblázat megtalálható az 52. CD Magazin/Tv könyvtárában.

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

Héjprogramozás Linux alatt – a sed használata (6. rész)

Az előző részben az awk nyelvvel ismerkedtünk meg, amellyel számos különböző szövegátalakítási és matematikai műveletet végezhetünk el a héjprogramokban. Ebben a részben egy másik szövegkezelő eszközről, a sed segédprogramról lesz szó.

A sed (stream editor) tulajdonképpen egy programozható szövegszerkesztő, ami a szabványos bemenetére érkező szöveget „röptében” képes átalakítani (természetesen fájlból is tud olvasni).
Működésének lényege, hogy a feldolgozandó szöveget soronként egy átmeneti tárba, az úgynevezett mintatérbe olvassa be, szabályos kifejezések alapján megkeres benne bizonyos részeket, majd elvégzi rajtuk az egybetűs parancsok formájában megadott műveleteket.
Bár a sed így, a XXI. század hajnalán sokak számára igencsak „fapadosnak” tűnhet, ha megtanulunk bánni vele, csodákra képes. Kezdők számára a legriasztóbb vonása a szabályos kifejezések használata, ami kétségkívül a Unix egyik legnehezebben megtanulható részét képezi. Ezzel kapcsolatban biztatásul is csak annyit mondhatok, hogy feltétlenül érdemes elsajátítani. A szabályos kifejezések teljes ismertetésére ez a rövid cikk messze nem elég, ugyanakkor tömör és világos leírást találhatunk róluk gyakorlatilag valamennyi, a Unix operációs rendszerről szóló tankönyvben.
A sed-et leggyakrabban egy másik programtól érkező kimenet feldolgozására használjuk, valahogy így:

```
... | sed 'program' | ...
```

Az egyszeres idézőjelek között megadott 'program' itt a sed saját nyelvén írt szövegfeldolgozási utasítássorozatot jelenti, ami egybetűs parancsokat és szabályos kifejezéseket tartalmaz. Ez a program több sorban több feldolgozási lépést is tartalmazhat, amelyek a megadás sorrendjében mennek végbe. (A későbbi utasítások már a korábbi átalakítások eredményét kezelik, és nem az eredeti szöveget.) A sed programok egy sora legáltalánosabb formájában a következőképpen néz ki:

```
<cím1>,<cím2> parancs
```

Itt <cím1> és <cím2> egy-egy szám vagy szabályos kifejezés lehet. Ha számot adunk meg, az a bemenet adott sorszámú sorának feldolgozását jelenti, ha szabályos kifejezést (ezt két / – perjel – közé kell zárni), akkor a program minden olyan sorra lefut, amelyre a kifejezés illeszkedik. Ennek megfelelően a

```
25 egybetűs_parancs
```

utasítás csak a bemenet 25. sorát fogja érinteni, míg a

```
/[0-9]/ egybetűs_parancs
```

programsor minden olyan szövegsort érinteni fog, amelyben legalább egy számjegy van.
A sed működése közben amolyan átfolyóként viselkedik, ami azt jelenti, hogy azokat a sorokat is kiküldi a kimenetére, amelyekkel semmi dolga nem akadt. A használható parancsokat *táblázatban* foglaltam össze. Ezek működéséről, illetve a szabályos kifejezések használatáról a megfelelő súgóoldalakon találhatunk bővebb ismertetést. (Mint korábban is említettem, kezdők számára erősen ajánlott valamilyen Unix-tankönyv böngészése is.)

A sed egybetűs parancsai

p	Kiíratás
d	Törlés
s	Helyettesítés
a	Hozzáfűzés
i	Beszúrás
c	A mintatér cseréje
y	Karakterek cseréje

Tizedes törtek

Ennyi bevezető után nézzünk egy egyszerű példát! Mint azt nyilván az olvasó is tudja, a magyar szövegekben előforduló tizedes törtekben a nyugaton szokásos tizedespont helyett tizedesvesszőt használunk. Ugyanakkor gyakori, hogy egy számsorozat vagy egy táblázat egy program kimeneteként keletkezik, s így mégis pontokat tartalmaz. Írjunk most egy olyan héjprogramot, amelyik „magyarosítja” a tizedes törtek írásmódját!
Bár a feladat első látásra egészen egyszerűnek tűnhet, ne feledkezzünk meg róla, hogy egy szövegben sokféle helyen szerepelhet pont (például e mondat végén is). Az tehát biztosan nem jó megoldás, ha az összeset egyszerűen vesszőre cseréljük le.
Az első feladat a tizedespont egyfajta logikai meghatározása, aminek alapján megírhatjuk a behatárolásához szükséges szabályos kifejezést. Nos, miről is ismerszik meg egy jó családból való tizedespont? Két számjegy határolja, vagyis illeszkedik rá a

```
[0-9]\.[0-9]
```

szabályos kifejezés. Ezzel a lényeg tulajdonképpen meg is van, de akad itt még néhány nehézség. Csábító ugyan a gondolat, de a fenti mintát nem használhatjuk egy cserélő utasításban a következő módon:

```
sed 's/[0-9]\.[0-9]/,/g'
```

Az a baj ezzel a megoldással, hogy a csere a teljes illeszkedő részre vonatkozik, vagyis a tizedespontot megelőző és az azt követő számjegyet is törölni fogja (a sor végén szereplő „g” egy jelentésmódosító kapcsoló, ami az összes lehetséges illeszkedés cseréjét írja elő). Az sem teljesen jó megoldás, ha az előbbi szabályos kifejezést a feldolgozandó sor címzésére használjuk:

```
sed '/[0-9]\.[0-9]/ s/\./,/g'
```

Ilyenkor a helyettesítés a tizedes törtet is tartalmazó sorokban minden pontot érinteni fog. A helyes programnak a következőket kell tartalmaznia:
1. Megkeressük a két számjegy által határolt pontokat.
2. A pont mindkét oldalán megkeressük és megjegyezzük a leghosszabb, csak számjegyekből álló karakterláncot.
3. A két karakterláncot önmagával, a köztük levő pontot pedig vesszővel helyettesítjük.

Ennek a műveletsornak a következő `sed` program felel meg:

```
sed 's/\([0-9][0-9]*\)\.\([0-9]
➥[0-9]*\)/\1\,\2/g'
```

Itt bizony már látszik a szabályos kifejezéseknek a kezdő felhasználókat gyakran megbabonázó szépsége. Lássuk, miről is van szó. Valahol a minta közepénél felismerhető a keresett tizedespont (`\.`). Ennek mindkét oldalán kerek zárójelek között szerepel a `[0-9][0-9]*` szabályos kifejezés, ami emberi nyelvre lefordítva egyszerűen azt jelenti, hogy 'legalább egy számjegy'. (A szabályos kifejezésekben a * jelentésmódosító műveleti jel (operátor) „nullaszoros” illeszkedést is megenged, ezért kellett kétszer kiírni a `[0-9]` tartományt.) A kerek zárójelek hatására a `sed` nemcsak megkeresi a pont két oldalán található, csak számjegyekből álló karakterláncot (a tizedes tört egészrészét és törtrészét), hanem meg is jegyzi azokat két átmeneti tárban. Ezekre hivatkozunk a helyettesítésnél a `\1` és `\2` szimbólumokkal.
Kész programunk némi „szokásos körítéssel” kiegészítve a következőképpen fest:

```
 1: #!/bin/sh
 2: PROGRAMNEV=`basename $0`
 3: if [ $# -ne 1 ]
 4: then
 5:   echo "Használat: $PROGRAMNEV fájlnév"
 6:   exit 1
 7: fi
 8: if [ -f $1 ]
 9: then
10:   cat $1 | sed\
11:   's/\([0-9][0-9]*\)\.\([0-9]
      ➥[0-9]*\)/\1\,\2/g'
12: else
13:   echo "A fájl nem létezik!"
14:   exit 2
15: fi
```

Szűrőként működő héjprogramok

A cikk elején említettem, hogy a `sed` egyik legfontosabb tulajdonsága „átfolyóként” való működése, vagyis hogy a beérkező szöveget akkor is átereszti, ha semmilyen műveletet nem kellett rajta elvégeznie. Az előbbi héjprogramunk azonban sajnos úgy lett megszerkesztve, hogy általa a `sed`-nek ezt az értékes tulajdonságát elveszítjük. Amennyiben ilyen szűrőként működő tizedespont-kezelésre van szükségünk, bármikor megtehetjük, hogy a megfelelő helyen csak magát az előbb kidolgozott `sed` programot használjuk. Ez azonban érezhetően körülményes, és egyébként is, ki szokott kapásból emlékezni ilyen körmönfont megoldásokra?
Ez az a pont, ahol nagy hasznát vehetjük a `read` parancsnak, ami a héjprogram szabványos bementéről egysornyi adatot olvas be az utána megadott héjváltozóba. Ha az olvasás sikeres volt, visszatérési értéke igaz, ha nem, akkor hamis. Ezt az utóbbi tulajdonságát kihasználva egy ciklusban önműködően érzékelhetjük a bemenet végét, valahogy így:

```
1: #!/bin/sh
2: while read sor
3: do
4:   echo $sor | sed \
5:   's/\([0-9][0-9]*\)\.\([0-9]
     ➥[0-9]*\)/\1\,\2/g'
6: done
```

Ez a program pontosan ugyanúgy működik, mint az előbbi, de a feldolgozandó szöveget nem fájlból veszi, hanem a szabványos bemenetéről. Ha tehát mondjuk ***átfolyó.sh***-nak hívják, akkor a következőképpen kell használni:

```
cat feldolgozandó_szöveg.txt | átfolyó.sh
```

A nagy egyesítés

Számos olyan Unix-parancs van, amelyik a szabványos bemenetéről és egy parancssori kapcsolóként megadott fájlból egyaránt képes dolgozni. Ráadásul önműködően felismeri a pillanatnyi helyzetet, vagyis nem kell külön elmagyarázni neki, hogy mit akarunk. Esetünkben is sokkal célszerűbb volna ezt a viselkedésformát megvalósítani, mint két, valójában azonos feladatot ellátó programot írni. Íme az egyik lehetséges megoldás:

```
 1: #!/bin/sh
 2: # Szabványos bemenet feldolgozása
 3: if [ $# -eq 0 ]
 4: then
 5:   while read sor
 6:   do
 7:     echo $sor | sed 's/\([0-9]
        ➥[0-9]*\)\.\([0-9][0-9]*\)/\1\,\2/g'
 8:   done
 9: # Parancssori fájlok feldolgozása
10: else
11:   for nev
12:   do
13:     cat $nev | sed 's/\([0-9]
        ➥[0-9]*\)\.\([0-9][0-9]*\)/\1\,\2/g'
14:   done
15: fi
```

Ez a program mindenképpen működik, hiszen vagy a szabványos bemenetet (5–8. sorok), vagy a parancssorban megadott valamennyi fájlt (11–14. sor) dolgozza fel. (Bár a `for` ciklus feje kissé befejezetlennek tűnhet, valójában nem az. Ha nem adunk meg listát, a ciklus alapértelmezésként önműködően a parancssori kapcsolókon megy végig.) Ha egyáltalán semmit nem adunk meg neki, amivel működhetne, akkor tőlünk várja, hogy gépeljünk neki valamit, akár a többi, hasonló célú Unix-segédprogram.

Büki András (buki.andras@insilico.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.

A naplózott fájlrendszer, a biztonság és az ACL (15. rész)

A naplózott fájlrendszer után azokat a védelmi mechanizmusokat mutatjuk be, amelyek segítségével adatainkat megvédhetjük a rendszer többi használójától.

Az előző részben a „hagyományos” fájlrendszerekkel foglalkoztunk, amelyek jól működnek ugyan, de korszerűeknek semmiképp sem nevezhetők. Ennek oka a technológiai fejlődés érdekes aszimmetriája. Míg a processzorok sebessége nagyjából másfél évente megkétszereződik, a memóriák mérete pedig az idő előrehaladtával exponenciálisan növekszik (ez az úgynevezett Moore-szabály, ami 1965 óta nem vesztette érvényességét), addig a merevlemezek sebessége (pontosabban a fej pozícionálásának ideje) alig-alig változik.
A memóriák árának csökkenése és méretük gyors ütemben történő növekedése a gyorsítótárak növekedésével is együttjár. Ha a gyorsítótár mérete nő, több blokk fér bele, így több olvasási műveletet tudunk a lemezhez való fordulás nélkül kiszolgálni. Már a 90-es évek elején felmerült, hogy a jövőben az előreolvasási módszer (a blokkokat még azelőtt beolvassuk a gyorsítótárba, mielőtt valóban szükség lenne rájuk) a jövőben nem fog jelentős teljesítménybeli növekedéssel járni. Ebből adódóan a lemezhez való fordulások többsége írási művelet lesz.
Az igazi gond abból adódik, hogy a legtöbb rendszerben az írás mindig csak kis részekben zajlik, és ezek a kis részek általában egymástól távol helyezkednek el. Ez pedig sok pozícionálással, illetve a lemez sokszori forgatásával jár. Természetesen „gyorsítótárazhatnánk” itt is, de a legtöbb esetben (például új állomány létrehozásakor) a fájlleírókat (inode, illetve nem Unix-rendszerekben olyan blokkokat, amelyek a fájlrendszer szempontjából alapvetők) is írnunk kell a lemezre. Az ilyen írásokat azonban célszerű azonnal elvégezni, különben elég egy áramszünet, és máris felborul fájlrendszerünk egységessége.

Naplószerkezetű fájlrendszer

Szükség volt hát egy olyan fájlrendszerre, ami a mai körülményekhez edződött, azaz a „hagyományos” fájlrendszerekkel szemben sok kis írásnál sem veszít a hatékonyságából. Ez az úgynevezett naplószerkezetű fájlrendszer, az LFS (Log-structured File System). A különbség azonban nem a fájlok és a könyvtárak megvalósításában mutatkozik – itt is megtalálhatjuk a fájlleírókat és a fájlok tartalmának elérésében sincs komolyabb változás –, az eltérés a blokkok elhelyezkedésében rejlik.
A „hagyományos” fájlrendszerek esetében megszokhattuk, hogy a fájlrendszer szerkezetét leíró blokkok (Unix esetében ezek a fájlleírók) mindig egy kitüntetett helyen találhatóak a lemezen. Az LFS szakít ezzel a „hagyománnyal”. A lemezen nem azt tárolja, hogy mik találhatók az egyes állományokban, hanem azt, hogy milyen műveletek történtek a fájlrendszerrel. A lemezt tehát egy naplónak tekinti, amiben feljegyzi a fájlrendszerben történt változásokat.
Az LFS-t nem hiába nevezik naplószerkezetűnek, mivel a működése tényleg egy hétköznapi napló írásával egyezik meg. A napló általában egy könyv vagy egy füzet, amit az emberek arra használnak, hogy beleírják a nap történéseit.
A naplókat általában az elejéről kezdik el írni, és minden oldalra egy-egy nap története kerül.
Az LFS ugyanezt teszi: a memóriában összegyűjti a fájlrendszerben történő változásokat. Ha például megváltoztatjuk egy blokk vagy egy fájlleíró tartalmát, akkor az úgy, ahogy van, bekerül ebbe a memóriarészbe. Ezeket a függőben lévő írásokat úgynevezett bejegyzésbe pakolja, ezt pedig egy az egyben kiírhatjuk a lemezre, azaz hozzáfűzhetjük a naplónkhoz.
Minden bejegyzés mérete kötött, általában 1 MB. Az, hogy egy bejegyzés mit tartalmaz, attól függ, hogy milyen műveletek történtek. Véletlenszerűen találhatók benne adatblokkok és fájlleírók is. Minden bejegyzés elejére kötelező azonban egy összefoglalót tenni, ami elárulja nekünk, mi is az, amit az adott bejegyzés tartalmaz. Ezt úgy képzelhetjük el, mintha valaki úgy írná a naplóját, hogy először címszavakban összefoglalja a nap legfontosabb eseményeit, majd részletesen kifejti azokat.
Mivel a naplót lineárisan írjuk a lemezre, a lehető leghatékonyabban tudjuk az írási műveleteket elvégezni. A fájlleírók azonban a lemezen össze-vissza, a napló különböző bejegyzéseiben helyezkednek el, így a fellelésük sokkal bonyolultabb, mint a „hagyományos” fájlrendszerek esetében – az LFS ezért egy táblázatnak is helyet szorít a lemezen, ami az összes fájlleíró pontos helyét megadja.
A naplónknak otthont adó füzet előbb-utóbb betelik. Ilyenkor érdemes elbaktatni a legközelebbi papírboltba, és venni egy újabb füzetet. Az LFS esetében a helyzet összetettebb, hiszen a felhasználók fenntartással kezelnek egy olyan rendszert, ami időnként új merevlemez beszerzésére szólítaná fel őket.
Ám míg a hétköznapi naplók bejegyzései bármikor hasznunkra lehetnek, az LFS régebbi naplóbejegyzései sok felesleges dolgokat tartalmazhatnak, ilyenek az azóta már letörölt vagy felülírt állományok adatblokkjai.
Ennek következtében a háttérben mindig futnia kell egy úgynevezett takarító folyamatnak, aminek az a dolga, hogy ezeket a felesleges adatokat kiradírozza a naplóból, helyet szabadítva így fel az újabb naplóbejegyzéseknek. A gyakorlatban ez úgy működik, hogy a takarító a napló elejéről (tehát az első bejegyzéstől) elindul, s azokat az adatokat gyűjti össze egy újabb bejegyzésbe, amelyek még mindig érvényesek, ezután kiírja őket a napló végére. Ezt követően ezeket a bejegyzéseket felszabadítja, tehát ha betelt a merevlemez, az LFS ismét igénybe veheti őket. (Ha nagyon menőnek szeretnénk tűnni, akkor úgy kellene fogalmaznunk, hogy az LFS a lemezt egyfajta cirkuláris pufferként használja: az új bejegyzések a napló végére kerülnek, míg a takarító folyamat a napló elejéről csíp le egy-egy bejegyzést. A dolog azért ennél sokkal körmönfontabb, a fájlleírók helye ugyanis a takarítás közben megváltozik, így frissíteni kell egyrészt a fájlleírók helyét megadó táblázatot; másrészt – ha a kérdéses fájlleíró közvetett volt, lásd sorozatunk előző részében, akkor – más csomópontokat is.)
Az LFS tehát hatékonyabban kezeli a sok kis véletlenszerű írást, mint a „hagyományos” fájlrendszerek, ennek

azonban megvan az ára: bonyolódik a fájlok tartalmának elérése, illetve a takarításkor rendkívül sok felügyeleti tevékenység elvégzésére kényszerülünk. Emiatt felvetődhet a kérdés, hogy mindent egybevéve az LFS hatékonyabb-e, mint a többi fájlrendszer. A mérések azt mutatják, hogy ennek ellenére is érdemes használni az LFS-t. Hiszen a processzor és a memória ma már sokkalta gyorsabb, mint régen, így hiába több a munka, mégis nagyobb teljesítményt érhetünk el az LFS-sel, ha sok kis véletlenszerű írási műveletnek kell eleget tennünk. Mi a helyzet azonban a nagyméretű írásoknál, illetve az olvasásnál? A mérések szerint semmivel sem marad el a „hagyományos" fájlrendszerektől. Fontos, hogy semmiképp se keverjük össze az LFS-t az úgynevezett naplózó (journaling) fájlrendszerekkel – ez két teljesen különböző dolog. A naplózó fájlrendszereket (például `ext3`) azért fejlesztették ki, hogy egy váratlan rendszerleállás után a fájlrendszer egységességét gyorsan helyre lehessen hozni. Az `fsck` ugyanis akár több órán keresztül is elszöszmötölhet a fájlrendszerrel, ami például egy kiszolgáló esetében elég kellemetlen lehet. A gyors helyreállítást úgy éri el, hogy minden írási műveletet először csak egy adatbázisba (a naplóba) ír bele, majd ha a lemeznek már nincs jobb dolga, a művelet csak akkor hajtódik végre valóban. Ha a rendszer esetleg összeomlana, mielőtt a naplóban feltüntetett műveletek befejeződtek volna, akkor a rendszer újbóli indulásánál elég csak a naplót megvizsgálni, hogy volt-e félbemaradt művelet. Ha volt, akkor azt fél másodperc alatt el lehet végezni, és nem kell az egész fájlrendszer összes fájlleíróját megvizsgálni. Ez ugyan nagyon hasznos dolog, csak épp semmi köze sincs a naplószerkezetű fájlrendszerhez. Linuxhoz sajnos nem ismerünk megbízhatóan használható LFS-megvalósítást. Bár számos projekt célul tűzte ki a megvalósítását, a legtöbb a fejlesztés korai szakaszában abbamaradt.
A FreeBSD viszont támogatja az LFS-t.

A védelem kérdése

Az operációs rendszernek nemcsak adataink tárolásáról, de védelméről is gondoskodnia kell. Pontosabban arról, hogy meghatározható legyen, hogy ki mihez férhet hozzá, illetve milyen műveleteket hajthat végre például egy állományon (esetleg csak olvashassa, de ne írhassa felül stb.). Ennek módját védelmi mechanizmusnak nevezzük, ami szigorúan technikai feladat.

Ez az az elv, amelynek alapján az operációs rendszer a védelmi adatokat tárolja. Egy védelmi elv akkor jó, ha egyrészt nem játszható ki. (Például nincs benne olyan kiskapu, aminek a segítségével a felhasználó képes olyan dolgot megtenni, amit neki nem lenne szabad. Ha valamiféle programhiba miatt történne mindez, az nem számít. Attól, hogy a megvalósítás rossz, az elv még jó lehet.) Másrészt szabadságot is kell biztosítania ahhoz, hogy úgy alakíthassuk ki az erőforrások védelmét, ahogy az nekünk tetszik. Nem jó az a védelmi mechanizmus, amelyik csak tiltást és engedélyezést valósít meg, mivel ennél sokkal finomabb árnyalatokra is szükség lehet. Például előfordulhat, hogy jó lenne, ha Józsi csak olvashatna az adott fájlból, míg Pisti csak írhatna bele, és Bélának semmiféle jogosultsága nem lenne hozzá.

Védelmi mechanizmusok

Először is mi az, amit egy rendszerben védenünk kell? Védenünk kell magát a gépet: ide soroljuk a processzort (például egy folyamat ne terhelhesse korlátlanul), a memóriaszakaszokat (az egyik folyamat ne írhasson a másik memóriaterületére) vagy a nyomtatót (ne minden felhasználó használhassa, illetve a nyomtatási sorból ne törölhesse ki más nyomtatnivalóját). Óvnunk kell továbbá a programot is – ez alatt a folyamatokat, az állományokat és az adatbázisokat értjük. A védelmezendő eszközök összességét és a programokat együttesen objektumoknak nevezzük.
Az objektumok fontos tulajdonsága, hogy nevük van. A név a rájuk való hivatkozást szolgálja, ezért egyedi, azaz nem lehet még egy ilyen nevű objektum a rendszerben. Minden objektumhoz tartozik még egy úgynevezett műveleti lista, ami azokat a műveleteket tartalmazza, amelyeket az adott objektumon végre lehet hajtani. Egy állomány esetében ilyen művelet lehet az olvasás, írás, végrehajtás, átnevezés stb.
Azzal, hogy erőforrásainkat objektumokba csoportosítjuk, pontosan meghatározhatjuk, hogy mely folyamatok mit csinálhatnak, illetve mit nem csinálhatnak az adott objektummal. Ezek a folyamat adott objektumra érvényes jogosultságai. A védelmi mechanizmus tehát nem más, mint az a módszer, aminek a segítségével megtiltható, hogy egy folyamat elérhesse azokat az objektumokat, amelyekhez nincs jogosultsága.
A dolog azonban nem merül ki ennyiben, ugyanis léteznek olyan objektumok, amelyekhez a folyamat ugyan hozzáférhet, de nem hajthatja rajta végre az összes műveletet. Például olvashat egy állományt, de írni nem tud bele.
Ebben idáig még nincs semmi misztikus, teljesen hétköznapi dolog, hogy egyes állományokat olvashatunk, de nem írhatunk, vagy fordítva. Most azonban egy kicsit „vadabb" fogalmat vezetünk be azért, hogy könnyebben megérthessük, milyen a jó védelmi mechanizmus. De nem kell megijedni, ez sem lesz teljesen ismeretlen, inkább csak szokatlanabb megvilágításba helyezi a dolgokat. Ez a fogalom pedig a védelmi tartomány, ami alatt egy objektumot és a rá vonatkozó jogosultságokat értjük.
A jogosultságok alatt most azokat a műveleteket fogjuk össze, amelyeket egy folyamatnak végre szabad (azaz jogában áll) hajtania.
A folyamatok jól meghatározott védelmi tartományokban futnak. Hogy melyik objektumhoz férhetnek hozzá, illetve melyik objektumon milyen műveleteket hajthatnak végre, az attól függ, hogy éppen melyik védelmi tartományban tartózkodnak. A legtöbb rendszerben a folyamatok futás közben átugorhatnak az egyik védelmi tartományból a másikba. Ennek rendje és módja minden típusú rendszerben más és más.
Minden folyamat rendelkezik egy olyan jellemzővel, amelyik egyértelműen meghatározza a pillanatnyi védelmi tartományt. A Unix esetében ez az `uid` (User ID, felhasználó azonosító) és a `gid` (Group ID, csoportazonosító). Ezekkel a korábbi részek során már találkozhattunk. A Unix minden felhasználóhoz, illetve csoporthoz egy egyedi számot rendel, ami alapján azonosítja azt. A felhasználói név csak a felhasználók számára fontos (mivel egy nevet könnyebben meg lehet jegyezni, mint egy számot), a rendszer számára azonban minden felhasználó csak egy szám.
A folyamat `uid`-je és `gid`-je azt mondja meg, hogy melyik felhasználó, illetve csoport jogosultságaival bír.
Ezért az objektumokra vonatkozó jogosultságokat nem a folyamatokra, hanem a felhasználókra és a csoportokra határozzuk meg. Így az `uid` és a `gid` egyértelműen megadja az adott folyamat védelmi tartományát. Magyarán, ha két folyamat `uid`-je és `gid`-je megegyezik, akkor pontosan ugyanolyan jogosultságok vonatkoznak rájuk.
Nem biztos, hogy egy folyamat a futása során végig ugyanabban a védelmi tartományban marad, bizonyos esetekben valamiképpen át kell lépnie egy

TARTOMÁNYOK	File1	File2	File3	File4	Nyomtató1
1.	Olvasás, Írás		Olvasás		
2.		Olvasás, Írás, Végrehajtás		Olvasás	Írás
3.	Olvasás, Végrehajtás				Írás

1.

TARTOMÁNYOK	File1	File2	File3	File4	Nyomtató1	T.1	T.2	T.3
1.	Olvasás, Írás		Olvasás					
2.		Olvasás, Írás, Végrehajtás		Olvasás	Írás	Belépes		
3.	Olvasás, Végrehajtás				Írás			

2.

másikba. Erre a legkézenfekvőbb példát a `SETUID`-es, illetve a `SETGID`-es programok nyújtják. A Unix lehetőséget ad arra, hogy az `EXEC` rendszerhívás segítségével elindított program a tulajdonosának, és nem az őt futtatónak a jogosultságával fusson. Ilyen például a `passwd` nevű program, amelynek segítségével a felhasználók megváltoztathatják a jelszavukat. Ehhez az kell, hogy a ***/etc/passwd*** (illetve a mai rendszerekben a ***/etc/shadow***) állományt írhassák. Ezt azonban – érthető okokból – kizárólag a rendszergazda teheti meg, tehát a `passwd`-nek mindenképp `SETUID`-esnek kell lennie. (Monolitikus rendszerekben akkor is megváltozik a védelmi tartomány, amikor a folyamat egy rendszerhívást hajt végre. Ebben az esetben a folyamat felhasználói módról magmódra vált. A magmódban alig-alig van olyan objektum, amelyhez ne férhetnénk hozzá korlátlanul, így nyilván a védelmi tartomány sem lehet ugyanaz, mint a felhasználói módban.)

Hogy mindez valóra válhasson, az operációs rendszernek gondoskodnia kell a védelmi tartományok tárolásáról, különben nem tudná eldönteni, hogy a folyamat végrehajthatja-e az adott műveletet. Hogy ez miként történjen meg, azt határozza meg a védelmi mechanizmus.

Védelmi tartományok nyilvántartása

A legkézenfekvőbb megoldásnak az tűnhet, ha az egészet egy táblázatban tároljuk (*1. ábra*). Minden oszlop egy-egy objektumnak, a sorok pedig a tartományoknak felelnének meg. Ezáltal a táblázat minden eleme az adott objektumra vonatkozó megengedett (vagy más néven legális) műveleteket fogja tartalmazni, amelyeket az adott folyamat végrehajthat az objektumon. Az ilyen táblázatot védelmi mátrixnak nevezzük. Vessünk egy pillantást a *2. ábrára* is! Itt a dolgon csavartunk azáltal, hogy magukat a tartományokat is objektumoknak tekintjük, csak a rajtuk értelmezhető művelet nem éppen az írás, az olvasás, illetve a végrehajtás, hanem a belépés. Ez a művelet azt jelenti, hogy az összes első tartományban lévő folyamat átlépheti ebbe (a jelen esetben a második) védelmi tartományba. Láthatjuk tehát, hogy ez a táblázatos módszer remekül alkalmazható azokban az esetekben is, amikor például `SETUID`-es programokat futtatunk.

A védelmi mátrixok egyszerűen használhatóak és könnyedén átültethetőek lennének a gyakorlatba is, a tartományok nyilvántartására azonban sehol sem használják. Ennek az az oka, hogy rendkívül sok objektum létezik, és közülük a legtöbbhöz a folyamatoknak szinte semmiféle jogosultságuk nincs. Ebből adódóan a védelmi mátrix hihetetlenül nagy és „szellős” lenne. Nem gazdaságos megoldás teljes egészében tárolni egy olyan táblázatot, amelyik nagy és majdnem üres. Sokkal célravezetőbb, ha a védelmi mátrixot soronként vagy oszloponként tároljuk, olyan módon, hogy csupán a nem üres elemeket jegyezzük meg.

Hozzáférést vezérlő lista (ACL)

Kezdjük az oszloponkénti tárolással. Mivel mi csak a nem üres elemeket kívánjuk tárolni, az egészet úgy is felfoghatjuk, mintha minden objektumhoz egy tartományokból álló listát rendelnénk. Ebben a listában azonban csak azokat a tartományokat tüntetjük fel, amelyeknek van valamiféle jogosultságuk az adott objektumra. Ha az 1. ábrán látható példát vesszük alapul, akkor a `File1` nevű objektumhoz tartozó listában az első és a harmadik tartomány szerepelne. Ha ehhez a listához hozzávesszük a tartományokhoz tartozó elérési módot is (tehát azoknak a műveleteknek a halmazát, amelyet a kérdéses objektummal elvégezhetünk), akkor azt ACL-nek (Access Control List, azaz hozzáférést vezérlő listának) nevezzük.

Erre láthatunk egy példát a *3. ábrán* (megint csak az 1. ábrából kiindulva). Mivel (Unix esetén) a védelmi tartományokat az `uid` és a `gid`, azaz a felhasználó és annak csoportja határozza meg, a lista minden eleme három részből tevődik össze: a felhasználó, a csoport és az objektumon végrehajtható műveletek.

Az ACL ennél sokkal többet rejt magában. A *4. ábrán* újabb példát láthatunk ACL-ekre. Semmi akadálya sincs annak, hogy egy hozzáférést függetlenítsünk a `uid`-től vagy `gid`-től. Példánkban ezt a * (csillag helyettesítőkarakter) jelöli. Ahol ez található az `uid`, illetve a `gid` helyett, ott nem érdekes, hogy az adott folyamat melyik `uid`-be, illetve `gid`-be tartozik.

Az ACL igazi előnye azonban az utolsó sor esetében mutatkozik meg. Könnyedén megtehetjük, hogy valakit anélkül zárjunk ki egy bizonyos jogosultságból, hogy a vele egy csoportban lévő felhasználókat a dolog érintené. Az itt leírtakkal egy gond akad, hogy a Unix nem használ ACL-t, illetve mégis, de másmilyet, mégpedig egy kilenc bites „változatot”. Ez alatt azt értjük, hogy minden objektumhoz (a gyakorlat nyelvére átültetve: minden állományhoz) három darab „rwx bitet” rendel: egyet a fájl tulajdonosához, egyet a csoportjához és egyet mindenki máshoz. Kár tagadni: ez nagyon messze áll egy teljes értékű ACL-es megoldástól. Ez a Unix egyik komoly hátránya. Noha a különlegesebb eseteket kivéve ennyivel is elboldogulunk, de

TARTOMÁNYOK	File1	File2	File3	File4	Nyomtató1
1. Józsi, tanuló	Olvasás, Írás		Olvasás		
2. Pityu, tanár		Olvasás, Írás, Végrehajtás		Olvasás	Írás
3. Samu, tanuló	Olvasás, Végrehajtás				Írás

3.

TARTOMÁNYOK	File5	File6	File7
1. Józsi, tanuló	Olvasás, Írás		Olvasás
2. Pityu, tanár		Olvasás, Írás	
3. Samu, tanuló			

4.

TARTOMÁNYOK	File1	File2	File3	File4	Nyomtató1
1.		Olvasás, Írás, Végrehajtás			
2.				Olvasás	
3.					Írás

5.

ha sok (több száz vagy ezer) felhasználónk van, akkor a felügyelet rendkívül költséges feladat. Ilyenkor érdemes elgondolkodni, nem járnánk-e jobban egy olyan operációs rendszerrel (például VMS), ahol egy kicsit „finomabban” is beállíthatjuk az egyes objektumokra vonatkozó jogosultságokat.
Az ACL-nek akad azonban hátránya is: ha egy objektum ACL-ét megváltoztatjuk, akkor az új „jogszabály” még nem fog azokra a folyamatokra vonatkozni, amelyek már megnyitották az adott állományt.

Képességi listák

A másik módszer a védelmi mátrix soronként történő tárolása. Ez tulajdonképpen az ACL fordítottja: nem azt tároljuk, hogy egy objektummal ki mit csinálhat, hanem azt, hogy ki melyik objektummal mit csinálhat. Némileg szakszerűbben megfogalmazva: a futó folyamatokhoz rendeljük hozzá az általa elérhető objektumokat a megengedett műveleteivel együtt (azaz amit jogosultsága van végrehajtani). Ez a képességi lista, amelynek elemei a képességek.
Tekintetünket szegezzük az *5. ábrára*, amelyen egy képességi listára láthatunk példát. Fontos, hogy a képességi lista az összes olyan műveletet magában foglalja, amit a folyamat végrehajthat. Annak sincs semmi akadálya, hogy a képességi listákra is objektumként tekinthessünk, amelyek más képességi listába (ez az úgynevezett tartománymegosztás) felvehetők.
Hogy egy folyamat egy rendszerben mit tehet meg és mit nem, az kizárólag a képesség listájának tartalmától függ. Könnyen belátható, milyen súlyos következménnyel járna, ha a folyamat némi ravaszkodás árán átírhatná a saját (vagy éppen egy másik folyamat) képességi listáját. Ilyesmit nem szabad megengedni, ezért a képességlistákat úgy kell tárolnunk, hogy a folyamatok még véletlenül se babrálhassanak vele. Erre több mód is kínálkozik: a legkézenfekvőbb az, hogy az operációs rendszer memóriaterületére tesszük, és a folyamatok a sorszámukkal hivatkoznak a képességekre. Kényelmesebb azonban, ha hardveresen oldjuk meg a feladatot, igaz, ehhez egyedi szolgáltatásra van szükség, ami elég kevés kiépítésben létezik. (Az eszközből történő megvalósítás úgy működne, hogy minden memóriaszóhoz egy bit lenne rendelve, ami azt mondja meg, hogy az adott szó tartalmaz-e képességet vagy sem.)
A leghatékonyabb megoldás azonban az lenne, hogy a képességlistát a folyamat saját területén tárolnánk. A védelmét úgy szavatolnánk, hogy a rendszer titkosítaná egy olyan kulccsal, amit kizárólag csak ő ismer. Kellően erős titkosítás használatakor nem fenyegetne minket az a veszély, hogy esetleg valamelyik folyamat feltöri, viszont az is igaz, hogy sokkal erőforrás-igényesebb megoldás, mint az előző kettő.
A biztonság mellett fontos kérdés még a jogosultságok beállítása, azaz miként lehet megadni, hogy ki mihez férhessen hozzá. Erre az a megoldás, hogy minden objektum rendelkezik úgynevezett általános jogosultságokkal, amelyek szintén kiadhatók a többi objektumfüggő jogosultságok (olvasás, írás, végrehajtás stb.) mellé. Az általános jogosultságok egyike lenne például egy új képesség létrehozása vagy éppen törlése.
A képességi listás megoldás elsősorban osztott rendszerek esetében lehet kényelmes, de ennek is megvan a maga ára. Például sokkal nehezebb feladat egy jogosultság visszavonása. Ennek az az oka, hogy itt az objektumokhoz tartozó képességek a lemezen összevissza találhatók, míg az ACL esetében csupán egyetlenegy módosításra volt szükség.
Most már tudjuk, hogyan tárolja a rendszer azt, hogy melyik felhasználó mihez férhet hozzá. Ám a biztonság kérdése ennél sokkal messzebbre nyúlik. A következő részben éppen ezért a felhasználók azonosításával kapcsolatos kérdésekről, a jelszavakról, programhibákról, vírusokról és férgekről, rejtett csatornákról lesz szó, és még sok minden másról.

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.

Az XML és a Perl (2. rész)

Amikor a nyelv találkozik az értelmezővel…

Amikor egy nyelv értelmezéséről beszélünk, arra az elemzésére gondolunk, amikor a nyelv szabályait követve apró részekre bontjuk fel a bemeneti adathalmazt. Az értelmezés alapvetően szükséges minden olyan program esetén, amelyik strukturált adatot használ bemenetként. Az előző részben az egyszerű adatnyelvek között említést tettem egy olyan módszerről, amelyik vesszővel elválasztott mezőket tartalmazó sorokból álló állományt használ adatforrásként. Perlben az utóbbit valami ehhez hasonló módon értelmeznénk:

```
while (<INPUT>) {
       # @mezok tomb felhasznalasa
       @mezok = split /,/;

}
```

A fenti kód azonban csak ráhúzza a szerkezetet az olvasott állományban tárolt karaktersorozatra. Ebben az esetben a szerkezet implicitnek nevezhető. Az az adat, hogy a rekordok első mezője mondjuk valakinek a vezetékneve, míg a második a keresztneve, explicit módon nem derül ki az állományból. A programnak kell tudnia, hogy melyik micsoda.

Hogyan ne értelmezzünk XML-t?

Kezdő, sőt még komolyabb Perl-programozók is gyakran beleesnek abba a hibába, hogyha az adatforrás egy HTML-dokumentum vagy egy XML-alkalmazás, azonnal sorról sorra értelmezik és szabályos mintaillesztéssel keresik meg a címkéket. Ez nem túl jó módszer szerkesztett jelölőnyelvek esetében! Először is a sortöréseknek semmi jelentőségük nincs az SGML/XML-alkalmazások többségénél, mindössze a szerkesztés kényelmessé tételét szolgálják.
Másodszor, aki olvasta a Perl GYK-t, az tudja, hogy a Perl mintaillesztési kifejezések nem használhatók egymásba ágyazott zárójelek esetén (ugye, mind olvastátok a GYK-t?). „Mi köze van ennek az XML-hez?" – kérdezheted. XML-t értelmezni egy az egyben egymásba ágyazott zárójelek sokaságának értelmezését jelenti.
Harmadszor az XML-ajánlás engedélyezi logikailag egy dokumentum fizikai értelemben több részben történő tárolását.
Az SGML, amire a HTML is épül, ugyanazt a beszúrási módszert teszi lehetővé, csak sajnos igen kevés böngésző ismeri. Senki sem szeretné, hogy XML-alkalmazása ugyanilyen nehézségektől szenvedjen. A beszúrás módszere temérdek egyszerű, egyúttal unalmas kódot követel a programozótól.

Akkor hogyan értelmezzünk XML-t?

Az eddig leírtak figyelembevételével beláthatjuk, hogy XML-értelmezőt írni nem egy leányálom. A W3C egyik eredeti szándéka az XML kapcsán az volt, hogy egy diplomás informatikus egy hét alatt képes legyen megírni egy értelmezőt. Úgy néz ki, ez azoknak a céloknak az egyike volt, amit nem sikerült elérni, s aminek a megvalósítását az XML fejlesztésével megjelent új lehetőségek, például a névtér használata, még lehetetlenebbé tették.
Egy XML-értelmező felépítésének összetettsége ennek ellenére sem fogott vissza több tehetséges, magányos programozót és csoportot – ők megírták a saját értelmezőjüket. Ezek közül igen sok szabadon hozzáférhető. Így majdnem mindenki, aki XML-t használ bemenetként, egy előre megírt értelmezőt használ, ami az összes piszkos munkát elvégzi. A Perl világában ez olyan értelmezők használatát jelenti, amelyek modulok formájában elérhetőek.

Az értelmezők típusai

Az XML-dokumentumot az összes értelmező az alábbi összetevőkre bontja:

- elemek,
- egy tulajdonság-érték párokból álló lista minden elemhez (ez üres is lehet),
- az elemek tartalmát jelentő karakterek (szöveg),
- feldolgozói utasítások, amelyek egy bizonyos programnak szólnak, és nem az adatot írják le, XML-ben ezek `<?` és `?>` között találhatók; egy feldolgozói utasítás tartalma egy tetszőleges név, amit célnak is hívunk, és azt az alkalmazást jelöli, amelynek az utasítás szól; ezt egy egyenlőségjel követi, majd egy karakterlánc, amit már az alkalmazás fog feldolgozni, és
- megjegyzések, azaz olyan szövegek, amelyek `<!--` és `-->` között találhatók.

Valójában egy XML-értelmezőnek nem kötelessége az alkalmazásnak a megjegyzéseket átadnia, de nagyon sok megteszi. Ezért ne számíts erre a viselkedésre, ne akarj így utasításokat átadni egy programnak. Erre használd a feldolgozói utasításokat.
Az XML-értelmezők osztályozásakor két független szempontot kell figyelembe vennünk: vannak ellenőrző és nem ellenőrző, illetve folyamalapú és faalapú értelmezők.

Ellenőrző és nem ellenőrző értelmezők

Az előző cikkben megemlítettem, hogy az SGML-értelmezőknek a dokumentum helyes értelmezéséhez szükségük van a jelölőnyelv nyelvtanára, a DTD-re (Document Type Definition). A DTD írja le a használható elemtípusokat, az ezekhez tartozó tulajdonságokat (attributum), ha vannak, és azt, hogy hogyan lehet az egyes elemeket egymásba ágyazni. A HTML DTD-je többek között meghatározza, hogy a `<TD>` elem kizárólag a `<TR>`-en belül helyezkedhet el.
Az XML tervezésének köszönhetően egy jól formázott XML-dokumentumot DTD nélkül is értelmezni lehet. A jól formázott XML-dokumentummal szembeni követelmények az XML-ajánlásban találhatók. Így minden elemnek van kezdő és záró címkéje, az elemek egymásba ágyazhatók, de nem nyúlhatnak át egymáson, és minden dokumentum csak egy gyökérelemet tartalmazhat.
Bár a DTD nem kötelező egy XML-dokumentumban, egy ellenőrző értelmező használhatja a DTD-t arra, hogy megbizo-

nyosodjon az XML alkalmazásszabályainak betartásáról. Egyfajta séma ez – ha megszeged, az ellenőrző értelmező hangosan tiltakozik. Ez igen hasznos lehet akkor, ha külső forrásból származó XML-dokumentumot kell használnod. Például a beszállítód XML-formátumban küldte el a számlát – az utóbbi esetben nem árt ellenőrizni, hogy a megfelelő elemek a megfelelő sorrendben szerepelnek-e. A DTD továbbá meghatározhat alapértelmezett értékeket bizonyos tulajdonságokhoz, amit egy ellenőrző értelmező felhasználhat, ha az adott elemnél ez nincs megadva.
Egy nem ellenőrző értelmezőnek viszont nincs másra szüksége, mint hogy a dokumentum jól formázott legyen. A nem ellenőrző értelmezők sokkal egyszerűbbek ellenőrző társaiknál, emiatt a legtöbb ingyenes értelmező nem ellenőrző. A nem ellenőrző értelmezők kielégítő megoldást jelenthetnek, ha a dokumentum saját forrásból származik, vagy annyira összetett nyelvtana van, hogy nem lehet DTD-vel leírni, és az alkalmazásnak kell eldöntenie, hogy helyes-e. Létezik szabadon hozzáférhető ellenőrző XML-értelmező modul Perlhez `XML::Checker` néven, de sajnos még nem mondható üzembiztosnak.

Folyamalapú és faalapú értelmezők

Egy értelmező alapvetően kétféleképpen tálalhatja az adatot az alkalmazásnak. Az egyik módszer az, hogy a dokumentum olvasása közben minden egyes alkalommal, amikor új összetevőre bukkan, egy jelzést küld az alkalmazásnak. A másik lehetőség, hogy az egészet beolvassa, majd egy, a dokumentum felépítésével összhangban álló faszerkezetet ad át a programnak. Az elsőt folyamalapú vagy eseményvezérelt értelmezőnek, míg a másodikat faalapú értelmezőnek hívjuk. Két fogalommal gyakran fogsz találkozni a SAX és a DOM kapcsán. A SAX (Simple API for XML) egy, az xml-dev levelezési lista tagjai által létrehozott nem hivatalos ajánlás arról, hogy egy folyamalapú értelmező hogyan kell, hogy beszélgessen az alkalmazással.
A DOM (Document Object Model) a W3C hivatalos ajánlása, amelyben azt írja le, hogy egy alkalmazás hogyan érheti el és módosíthatja egy dokumentum faszerkezetét.
Melyiket használd? Ez a feldolgozás természetétől és a dokumentum méretétől függ. Egy faalapú értelmezőnek az egész dokumentumot be kell töltenie a memóriába, ami nem kifejezetten hasznos nagy szótárak vagy más adatbázisok esetén. Egy folyamalapú értelmezővel átugorhatod azokat az elemeket, amelyek nem érdekesek számodra. Egy adott szócikkre történő keresés esetén kifejezetten az utóbbi módszer javasolt. Viszont ha olyan elemeket keresel, amelyek kapcsolatban állnak más elemekkel (például azokat az írókat keresed, akik egy témában legalább három cikket adtak le), egy faalapú értelmezővel sokkal könnyebben boldogulhatsz.
Érdemes megjegyezni, hogy egy faalapú értelmező használható egy folyamalapú tetején, illetve egy faalapú értelmező kimenete bejárható úgy, hogy folyamalapú felületen keresztül érje el az alkalmazás.

XML-értelmező modulok Perlhez

Jelenleg négy főbb XML-értelmező érhető el modulok formájában. Ezek mind szabadon hozzáférhetőek és letölthetőek a CPAN oldaláról (➲ http://www.cpan.org).

XML::Parser

Ez az összes értelmező nagypapája. ***James Clark*** C-ben írt Expat értelmezőjén alapul, és eredetileg nem más, mint ***Larry Wall*** írta. Jelenlegi karbantartója ***Clark Cooper***.
Az `XML::Parser` alapvetően folyamalapú, nem ellenőrző értelmező; lényegében függvényeket kell rendelnie az egyes eseményekhez. Az esemény kiváltásakor a megadott függvény meghívásra kerül.

XML::DOM

Ez a faalapú értelmező a W3C 1. szintű DOM-felületét valósítja meg. ***Enno Derksen*** írta; az `XML::Parser` képezi az alapját.

XML::Parser::PerlSAX

Ken MacLeod viszonylag új folyamalapú értelmezője a SAX-felületet valósítja meg. A felülete sajnos nem annyira perles, mint az `XML::Parser`-é.

XML::Grove

Ken MacLeod faalapú értelmezője. Míg az `XML::DOM` külön függvényhívásokat igényel a fa bejárásához, az XML::Grove olyan fát hoz létre, amelyik szokványos Perl-tömbműveletekkel használható. Emiatt gyorsabb lehet, ha sokszor kell bejárni a fát, mivel Perlben a függvényhívás meglehetősen erőforrás-költséges.
Ezek a modulok egytől egyig objektumközpontú felülettel rendelkeznek. Ha még nem barátkoztál meg az objektumközpontú programozással Perl alatt, itt az ideje, hogy átnézd a `perltoot(1)` és `perltootc(1)` kézikönyvoldalakat – Debian alatt a ***perl-doc*** csomag tartalmazza őket.

Következzen egy valódi alkalmazás!

Eleget beszéltünk arról, hogy mit lehet beszerezni, kezdjük el használni is! Az XML egyik lehetséges felhasználási módja – habár nem az egyetlen –, hogy leírja egy dokumentum szerkezetét, így a nézegető már emberi fogyasztásra alkalmas formában jelenítheti meg. Ebben a példában egy rendkívül egyszerű XML alapú jelölőnyelvet hozunk létre, utána egy hihetetlenül rövid programot írunk a dokumentum megfelelő formázására, illetve a tartalomjegyzék kiíratására.

Dokumentumunk jelölőnyelve

A dokumentum egy vagy több fejezetből áll, amelyek egymást követő bekezdéseket és beágyazott fejezetek tartalmaznak. Minden fejezetnek van egy címe, és minden bekezdés természetesen karakterekből épül fel. Az első feladatunk ezeket az állításokat XML-es fogalmakkal leírni. A dokumentum egyértelműen egy elem, nevezetesen a gyökérelem. Ez alapján a dokumentumunk így néz ki:

```
<dokumentum>
<!-- valami meg jon ide -->
</dokumentum>
```

Továbbá minden fejezetnek külön elemnek kell lennie, ezeket hívjuk `<fejezet>..</fejezet>`-nek. A fejezetcím lehetne külön elem a `<fejezet>` elemen belül, de ezúttal legyen a `<fejezet>` elem tulajdonsága. Végül a bekezdéseket a `<bekezdes>` elemmel fogjuk jelölni. Ezek szerint egy teljes dokumentum így fest:

```
<dokumentum>
     <fejezet cim="elso">
       <bekezdes>Ez az elso fejezet</bekezdes>
       <fejezet cim="elso alfejezet">
            <bekezdes>Ez az elso fejezet
elso alfejezete
            </bekezdes>
```

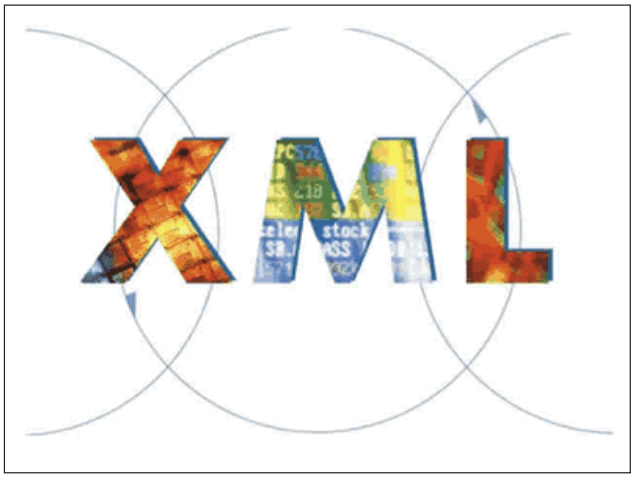
```
          <fejezet cim="elso al-alfejezet">
            <bekezdes>Ez az amit a cim is
sugall</bekezdes>
            <bekezdes>Ez meg a masodik
bekezdes</bekezdes>
          </fejezet>
          <bekezdes>Ez tovabbi szoveg
az elso alfejezethez. Jol nez ki, ugye?
          </bekezdes>
      </fejezet>
      <fejezet cim="masodik alfejezet">
          <bekezdes>Ez az elso fejezet
masodik
alfejezeteben szerepel.
          </bekezdes>
      </fejezet>
    </fejezet>
    <fejezet cim="masodik">
      <bekezdes>Ez a nagyon egyszeru program
az XML-
dokumentumok folyamalapu ertelmezeset
mutatja be. Beolvassa az allomanyt es
megfelelo sorszamozassal, bekezdesekre
tordelve talalja a tartalmat.
      </bekezdes>
    </fejezet>
</dokumentum>
```

Nézegetőprogramunk

Most egy olyan programot szeretnénk írni, ami a fentihez hasonló dokumentumot ekképp jeleníti meg:

```
1 elso

Ez az elso fejezet

    1.1 elso alfejezet

    Ez az elso fejezet elso alfejezete

        1.1.1 elso al-alfejezet

        Ez az amit a cim is sugall

        Ez meg a masodik bekezdes

    Ez tovabbi szoveg az elso alfejezethez.
Jol nez ki, ugye?

    1.2 masodik alfejezet

    Ez az elso fejezet masodik alfejezeteben
szerepel.

2 masodik

Ez a nagyon egyszeru program az XML-
dokumentumok folyamalapu ertelmezeset
mutatja be. Beolvassa az allomanyt es
megfelelo sorszamozassal,
bekezdesekre tordelve talalja a tartalmat.
```

Esetleg így:




```
1 elso
    1.1 elso alfejezet
        1.1.1 elso al-alfejezet
    1.2 masodik alfejezet
2 masodik
```

Figyeld meg, hogy XML-dokumentumunk sehol sem tartalmazza, hogyan kell sorszámozni, illetve bekezdésekre tördelni a szöveget. Kizárólag a fejezetek és bekezdések helye van megadva. Ez is azt a nézetet tükrözi, hogy a tartalmat és a kinézetet el kell választani egymástól – ez nagy előnye a strukturált jelölőnyelveknek.
Mivel a feldolgozás módszere nem igényel véletlen elérést a dokumentumban, a folyamalapú `XML::Parser`-t fogjuk használni.

```
#!/usr/bin/perl -w
use strict;
```

Ugye, mindig használod a `-w`-t és a `strict`-et?

```
use XML::Parser;
use Text::Wrap;
use Getopt::Std;

my ($indlevel,@sectnums,$parabuf,%opts);
```

A `$indlevel` a beágyazás mélysége, a `@sectnums` a fejezetszámokat tároló tömb, a `$parabuf` pedig mindig a pillanatnyi bekezdést tartalmazza.

```
getopts('c',\%opts);
```

Ha a `-c` kapcsoló meg van adva, akkor csak a tartalomjegyzéket írjuk ki.

```
die "Usage: ".$0." [-c] file\n"
➥unless @ARGV == 1;
my $p = new XML::Parser (Style => 'Stream');
```

Létrehozunk egy új értelmező objektumot, és meghatározzuk, hogy a `'Stream'` stílust akarjuk használni. A stílusokról bővebben az `XML::Parser` POD-jában olvashatsz.

```
$p -> parsefile ($ARGV[0]);
```

Az objektumnak átadunk egy állományt, és hagyjuk, hogy végezze a feladatát. Ezzel az utasítással a program véget ért. Az igazi érdekesség azonban az események kiváltásakor életbe lépő függvényekben rejlik:

```
sub StartTag {
    my ($expat,$eltype) = @_;
```

Amikor a `'Stream'` stílussal dolgozunk, az értelmező – akárhányszor csak meglát egy nyitócímkét – meghívja a `StartTag` függvényt. Az első átadott érték egy hivatkozás az értelmező objektumra, a második pedig egy karakterlánc a címke nevével. A `%_` a tulajdonság–érték párok listáját tartalmazza, míg a `$_` a teljes címkét magába foglalja a kisebb és nagyobb jelekkel együtt.

```
    if ($eltype eq "dokumentum") {
      $indlevel = -1;
      $sectnums[0] = 0;
      $parabuf = "";
```

Változóinkat kiindulási értékekkel látjuk el, amikor a gyökérelemmel találkozunk. Ezt egy `StartDocument` függvényben is megtehettük volna, ami akkor kerül meghívásra, amikor az értelmező hozzálát az elemzéshez.

```
    } elsif ($eltype eq "fejezet") {
      ++$sectnums[++$indlevel];
      $sectnums[$indlevel+1] = 0;
      print ' 'x(4*$indlevel),join
      ➥('.',@sectnums[0..$indlevel]),
      ➥" ",$_{cim},"\n";
      print "\n" unless $opts{c};
```

A `<fejezet>` típusú elemnél meg kell növelnünk a beágyazás mélységét és a fejezetszámot, illetve a következő fejezetszámot nulláznunk kell. Továbbá a címet is ki kell íratni a megfelelő bekezdéssel és számozással.

```
    } elsif ($eltype eq "bekezdes") {
```

Ebben a példában a bekezdések elején nem teszünk semmit, de egy kifinomultabb program esetén erre az ágra is szükség lehet.

```
    } else {
      die "Ismeretlen elem: ",$eltype,"\n";
    }
```

Mivel nem ellenőrző értelmezőt használunk, legalább egy kis ellenőrzésre szükség van az alkalmazás részéről. Ismeretlen elem esetén azonnal megszakad a program futása.

```
}

sub Text {
    tr/\n/ /;
    s/^\s+//;
    s/\s+$//;
    return if $_ eq "";
    $parabuf = $_;
}
```

Az értelmező a `Text` függvényt hívja meg, amikor karakterlánccal találkozik. A `$_` változó tartalmazza a szöveget a következő nyitó- vagy zárócímkéig, feldolgozói utasításig, illetve megjegyzésig. Egy XML-értelmezőnek az összes karaktert kötelessége átadni az alkalmazásnak, beleértve a szóközöket, a sortöréseket és a tabulátorokat. Ezért minden felesleges karaktert le kell vágnunk a szöveg elejéről és végéről. Figyelmen kívül hagyjuk azokat a karakterláncokat, amelyek például csak szóközből állnak. Ezek mindössze az XML-dokumentum könnyebb szerkeszthetőségét szolgálják. A sortöréseket szóközökké alakítjuk.
Egyedül a `<bekezdes>` elemen belüli szöveg érdekel minket. Ezért programunk a `<bekezdes>`-eken kívül minden karaktert szép csendben figyelmen kívül hagy.

```
sub EndTag {
    my ($expat,$eltype) = @_;
    if ($eltype eq "dokumentum") {
    } elsif ($eltype eq "fejezet") {
      --$indlevel;
    } elsif ($eltype eq "bekezdes") {
      my $ind = ' ' x (4*$indlevel);
      print wrap($ind,$ind,$parabuf),"\n\n"
      ➥unless $opts{c};
    }
}
```

Amikor a zárócímkébe ütközünk, munkánk véget ért. A bekezdéseket formázva kiíratjuk a képernyőre.

Mi vár rád a következő részben?

Ennyi volt e hónapra. A következő részben folytatjuk vizsgálódásunkat az XML-értelmezők körül, és közelebbről is megnézzük a faalapú értelmezők használatát. A fentebb szóba került összes példa megtalálható a CD-mellékleten.

Köszönetnyilvánítás

Szeretnék köszönetet mondani ***Eric Bohlman***-nak a hathatós közreműködésért.

Fülöp Balázs (admin@guardware.com)
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.

A Bricolage

A Salon és más népszerű webhelyek mögött meghúzódó tartalomkezelő rendszer egyszerűen használható saját webhelyünk írói és szerkesztői számára is.

Az elmúlt néhány hónapban bemutattunk egy pár Zope alapú tartalomkezelő rendszert. Természetesen nem a Zope az egyetlen versenyző a nyílt forrású tartalomkezelő rendszerek mezőnyében. Az egyik egyre jelentősebbé váló csomag a Bricolage, amelyet ***David Wheeler*** írt és tart karban, és amely a `mod_perl` csomagon és a PostgreSQL adatbázis-kezelőn alapul.

A Bricolage program használatához nem szükséges számítástechnikai ismeret. Igaz, hogy a program módosítása, illetve karbantartása nagy szakértelmet kíván, de az Apache webkiszolgálóval vagy a Perl nyelvvel dolgozó emberek általában programozók vagy rendszergazdák, míg a Bricolage felhasználói a webhely írói, szerkesztői és készítői.

A Bricolage a való világban is bizonyított. Az Apache és a Perl évekig küzdött azért, hogy bekerüljön a főáramba. A Bricolage már jó ideje a Salon magazin tartalomkezelő rendszere, és az eWeek, valamint a Register webhelyeknél is folyamatban van az átállás. Ráadásul a szakma képviselői közül, akik nem mindig nyilatkoznak kedvezően a nyílt forrású programokról, egyre többen próbálják ki és írnak bírálatot a Bricolage csomagról. A többség kitűnő programnak értékeli, amely felveszi a versenyt sok ezer dollárba kerülő, zárt kódú vetélytársaival.

Ebben a hónapban áttekintjük a Bricolage telepítését és használatba vételét. A következő pár hónapban sok szempontból megvizsgáljuk a programot, kitérve a különféle webhelytípusokhoz és a közléshez való testreszabásra.

Alapok

Minden tartalomkezelő rendszer magja egy adatbázis. A kereskedelmi tartalomkezelő programok általában az Oracle vagy a Microsoft SQL Server adatbázis-kezelőt használják. Számos nyílt forrású program, beleértve több PHP alapú tartalomkezelő rendszert is, a MySQL adatbázis-kezelőt használja. Ezzel szemben a Bricolage a PostgreSQL-t választotta háttértárolóként.

A PostgreSQL-t gyakran úgy emlegetik, mint „a másik” nyílt forrású adatbázis-kezelőt, pedig már régóta támogatja a tranzakciókezelést, lehetővé téve, hogy több lekérdezést vagy parancsot egy kötegbe fogjunk, és „mindent vagy semmit” alapon hajtsunk végre. A PostgreSQL olyan képességekkel is rendelkezik, amit a komoly adatbázis-felhasználók elvárnak, például nézetek, felhasználó által megadott függvények, részkiválasztások, egyesítések, idegen kulcsok és sértetlenség-ellenőrzés. A PostgreSQL támogatja a Unicode szabványt, ami egyre inkább fontos a többnyelvű webhelyek kezelésénél.

A Bricolage a PostgreSQL-t használja háttértárolóként, de magát az alkalmazást Perlben írták. Legalább két módszer ismeretes a kiszolgálóoldali webes Perl-programok futtatására. Az egyik a CGI, ami lassú, de biztonságos és hordozható, a másik a `mod_perl`, ami gyors, de a biztonsággal gondok lehetnek, és csak az Apache webkiszolgálóval működik. A Bricolage a `mod_perl` segítségével dolgozik, ami azt jelenti, hogy a kódja – Perl-modulok rendszere – csak egyszer fordul le, „Perl-opkódok” formájában a memóriában tárolódik, és onnan hajtódik végre minden alkalommal. Ez a megoldás azt eredményezi, hogy a Bricolage működése gyors lesz.

Mint már említettem, a `mod_perl` csak az Apache webkiszolgálóval működik. Bár folyamatosan dolgoznak a `mod_perl` átvitelén az Apache 2.x számára, a cikk írásának idején (2003 júniusa) még csak az Apache 1.x változatával működik együtt. Mivel az Apache 1.x több különálló folyamatot futtat, ahelyett hogy egy folyamaton belül több szálat indítana, nincs lehetőség arra, hogy igazi adatbázisgyűjtőt hozzunk létre a különféle gyermek HTTP-kiszolgálók számára. A létrehozott adatbázis-kapcsolat viszont életben tartható az Apache és a PostgreSQL között az `Apache::DBI` modul használatával. A Bricolage pontosan ezt teszi, így az adatbázis-kapcsolatot nem kell minden alkalommal újra létrehozni, amikor egy felhasználó lekér valamit.

A Bricolage az adatokat Perl/HTML-sablonok segítségével jeleníti meg a felhasználóknak. Seregnyi ilyen sablon van általában is a Perlhez, a `mode_perl`-hez pedig különösen sok. Régóta nagy rajongója vagyok a `HTML::Mason` modulnak, és az egyik ok, amiért ennyire lelkesedem a Bricolage programért, az az, hogy a kedvenc megoldásaimat – PostgreSQL, `mod_perl`, Apache és `HTML::Mason` – egy, a végfelhasználó számára jó alkalmazásba fogja össze.

Telepítés

A Bricolage telepítése nem egyszerű folyamat. Ez nem a Bricolage szerzőinek és karbantartóinak a hibája, inkább azért ilyen, mert a Bricolage rengeteg modult használ CPAN-ból (Comprehensive Perl Archive Network). Még ma sem megy olyan simán a telepítés, ahogy mennie kellene, de a dolog folyamatosan fejlődik, és az újabb változatokat egyre könnyebb telepíteni.

A legegyszerűbben úgy telepíthetjük a Bricolage programot, hogy miután a Perl, az Apache és a `mod_perl` – statikus modulként, nem Apache dinamikus megosztott objektumként (DSO) – telepítve van, a `Bundle::Bricolage` virtuális modult telepítjük a CPAN-ból. A Perl-modulokat úgy lehet telepíteni a CPAN-ból, hogy elindítjuk a CPAN-héjat a `perl -MCPAN -e 'shell'` paranccsal, és a parancssorban kiadjuk az `install Bundle::Bricolage` parancsot. Ha viszonylag új Perl-héjat használunk, és megadtuk a `PGINCLUDE` és `PGLIB` környezeti változókat, akkor minden modulnak a legújabb változata magától és gond nélkül letöltődik és telepítődik.

A folyamat hosszú és bonyolult lehet, ha hozzám hasonlóan belebonyolódsz a CPAN függőségeinek erdejébe. Nem minden CPAN-csomag adja meg világosan, hogy milyen más csomagoktól függ. Először az LWP-t és a `Bundle::CPAN`-t telepítettem a CPAN-héjból. Ezután megkíséreltem a `Bundle::Bricolage` telepítését (Red Hat 7.3-as rendszeren), de nem sikerült elsőre telepíteni a `Cache::Cache` modult, másodszorra viszont igen. A `DB_File` telepítése is sikertelen volt, mert nem volt telepítve a `db3-devel` RPM-csomag, és ez gondot okozott az `Apache::Session` telepítésénél, amely azután a `HTML::Mason` telepítését tette lehetetlenné, amitől a Bricolage

függ. Gondok voltak a ***libapreq*** telepítésével is (mert az Apache már futott ugyanazon a kapun), és az `XML::Parser` csomaggal is (mert az `expat-devel` RPM-csomag nem volt telepítve). Szerencsére egy CPAN-köteg telepítése után a rendszer kiírja, hogy mely csomagokat nem sikerült tisztán telepíteni. Megkísérelhetjük ismét a köteg telepítését (ekkor a CPAN közli, hogy mely modulok vannak már telepítve, és melyek még nem).
A `Bundle::Bricolage` nem a Bricolage moduljait telepíti, hanem csak azokat, amelyektől a Bricolage függ. Ha meggyőződtünk arról, hogy a `Bundle::Bricolage` telepítése sikeres volt, akkor töltsük le a legújabb Bricolage-tarcsomagot a Bricolage honlapjáról (➲ http://www.bricolage.cc), csomagoljuk ki, majd adjuk ki a `make` parancsot. A `Makefile` ellenőrzi, hogy a kötelező és választható Perl-csomagok közül melyek vannak telepítve, és megkérdezi, hogy telepíteni szeretnénk-e a hiányzókat. Ellenőrzi azt is, hogy a `mod_perl` statikus modulként (és nem DSO-ként) van-e fordítva, és hogy a PostgreSQL telepítve van-e. Végül néhány felhasználói nevet és jelszót kér, amelyek a Bricolage adatbázisának elkészítéséhez szükségesek, valamint telepíti a rendszerre a program `HTML::Mason` összetevőit.
A kérdések megválaszolása után a telepítést a következő parancs kiadásával indíthatjuk:

```
make cpan && make test && make install
```

Ez a parancs letölti és telepíti a szükséges modulokat a CPAN-ból, ellenőrzi a Bricolage telepítését, hogy minden rendben van-e, és ha igen, telepíti magát a Bricolage programot.
A Bricolage telepítése közben felmerülhetnek gondok, de az egész telepítés meglepően könnyű a rendszer bonyolultságához képest. A `Makefile` értelmes, előre kitöltött válaszokat javasol, kényszerít a „bric” PostgreSQL-felhasználó jelszavának megadására, és általában gondoskodik arról, hogy minden jól és könnyedén menjen.
A telepítés után kiadtam a `bric_apachectl` parancsot az Apache elindításához, a webböngészőmet a gyökérkönyvtárra állítottam, és megjelent az üdvözlő bejelentkezési képernyő. A telepítés sikerült! A Bricolage leírása javasolja, hogy azonnal változtassuk meg a rendszergazda jelszavát, ami ennek megfelelően ***change me now!*** (változtass meg!) értékre van alapból beállítva.
Itt említendő meg, hogy a Bricolage elég sokat használja az előugró ablakokat. Én személy szerint utálom ezeket ablakokat, még akkor is, ha nem hirdetési célból jelennek meg. Bár megértem, hogy a HTML-ben tervezett felhasználói felületek gyakran nem nélkülözhetik az előugró ablakok használatát, nem bánnám, ha a Bricolage kevesebbet használná ezt a megoldást. További zavaró tényező, hogy sok belső hivatkozás JavaScript-függő, így nem használhatom a kedvenc böngészőmben a középső egérgombot, hogy ne új ablakban, hanem új böngészőlapon jelenjen meg az oldal. A fenti apróságok nem sokat változtatnak azon a tényen, hogy a Bricolage igazán fantasztikus csomag.

A motorháztető alatt

A Bricolage több részből tevődik össze: adatmodell a PostgreSQL-ben, számos Perl-modul és sablon, amelyek a modulok által megszerzett adatokat jelenítik meg a felhasználók számára. Ez hasonló más, bonyolult adatbázis-hátterű rendszerekhez. Bár a Bricolage kevesebb feladatot kíván megoldani, mint az OpenACS, és nyilvánvalóan más módszereket használ célja elérésére, mégis nagyon hasonlít rá és azokra a rendszerekre, amelyek háromrétegű felépítést használnak.
Ha belenézünk a Bricolage csomag telepítőkönyvtárába, láthatjuk a rendszer által használt PostgreSQL adatbázis-meghatározásokat. A kezdő PostgreSQL-felhasználók meglepődhetnek azon, amit ebben az állományban látnak, például sorszámokat és feltételeket. A sorszám (sequence) egy különleges számobjektum a PostgreSQL-ben, amelynek az értékeit sosem lehet újra felhasználni. Leggyakrabban arra használják őket, hogy egyedi azonosítókat hozzanak létre egy rendszerben, különösen, amikor elsődleges kulcsként is felhasználják a programozók. Ha a PostgreSQL-ben egy oszlopot `SERIAL` típusúként határozunk meg az `INTEGER` helyett, akkor tulajdonképpen sorszámot hozunk létre. A PostgreSQL régebbi változataiban kényelmetlen volt a ***SERIAL*** oszlopot tartalmazó táblák törlése, először a táblát kellett törölni, azután a sorszámot, amely a ***SERIAL*** oszlophoz volt rendelve. A PostgreSQL 7.3-as megjelenésétől kezdve a tábla törlése magával vonja a ***SERIAL*** oszlopokhoz rendelt sorszámok törlését is.
A feltételek (constraint) lehetővé teszik, hogy az adatbázis visszautasítsa az olyan `INSERT` és `UPDATE` parancsokat, amelyek egy megadott tartományon kívül eső értéket próbálnak beállítani. Például a Bricolage megad egy media táblát, amelyben minden elem 1-től 5-ig terjedő fontossággal (priority) rendelkezik, az alapértelmezett érték 3. Az oszlop meghatározása így néz ki:

```
ooks priority NUMERIC(1,0)  NOT NULL
              DEFAULT 3
              CONSTRAINT ck_media__priority
              CHECK (priority BETWEEN 1 AND 5)
```

A feltételnek adhatunk nevet, ebben az esetben ez a név `ck_media_priority`. A névadás egyszerűbbé teszi a hibák megtalálását és javítását, ugyanis amikor ebbe az oszlopba érvénytelen értéket próbálunk beszúrni, a PostgreSQL kijelzi a megsértett feltétel nevét. Ez eléggé nagy segítség akkor, amikor az adatbázis-meghatározásban vagy az adatbázissal együttműködő alkalmazásban kell a hibát megtalálni.
Az is meglepő lehet, hogy kevés függvény lett meghatározva.

KAPCSOLÓDÓ CÍMEK

A legfőbb ismeretforrás a Bricolage honlapja: ➲ http://www.bricolage.cc. Ez az oldal tartalmaz hivatkozásokat a letölthető forráskódra (amely a ➲ http://SourceForge-on van), a leírásra és a hibajelentéseket és továbbfejlesztési kéréseket gyűjtő Bugzilla-rendszerre (bugzilla.bricolage.cc).

Sok Bricolage programmal foglalkozó levelezőlista van a SourceForge-on a ➲ http://sourceforge.net/projects/bricolage weblapon.

A mod_perl honlapja a ➲ http://perl.apache.org

A Mason-könyv honlapja (➲ http://www.masonbook.com) és a Mason honlapján (➲ http://www.masonhq.com) tudhatunk meg többet.

David Wheeler, a Bricolage főprogramozója és karbantartója szintén fenntart egy honlapot, amelynek címe ➲ http://david.wheeler.net. Vállalkozása, a Kineticode a ➲ http://www.kineticode.com címen érhető el.

Az Apache webkiszolgáló ➲ http://httpd.apache.org

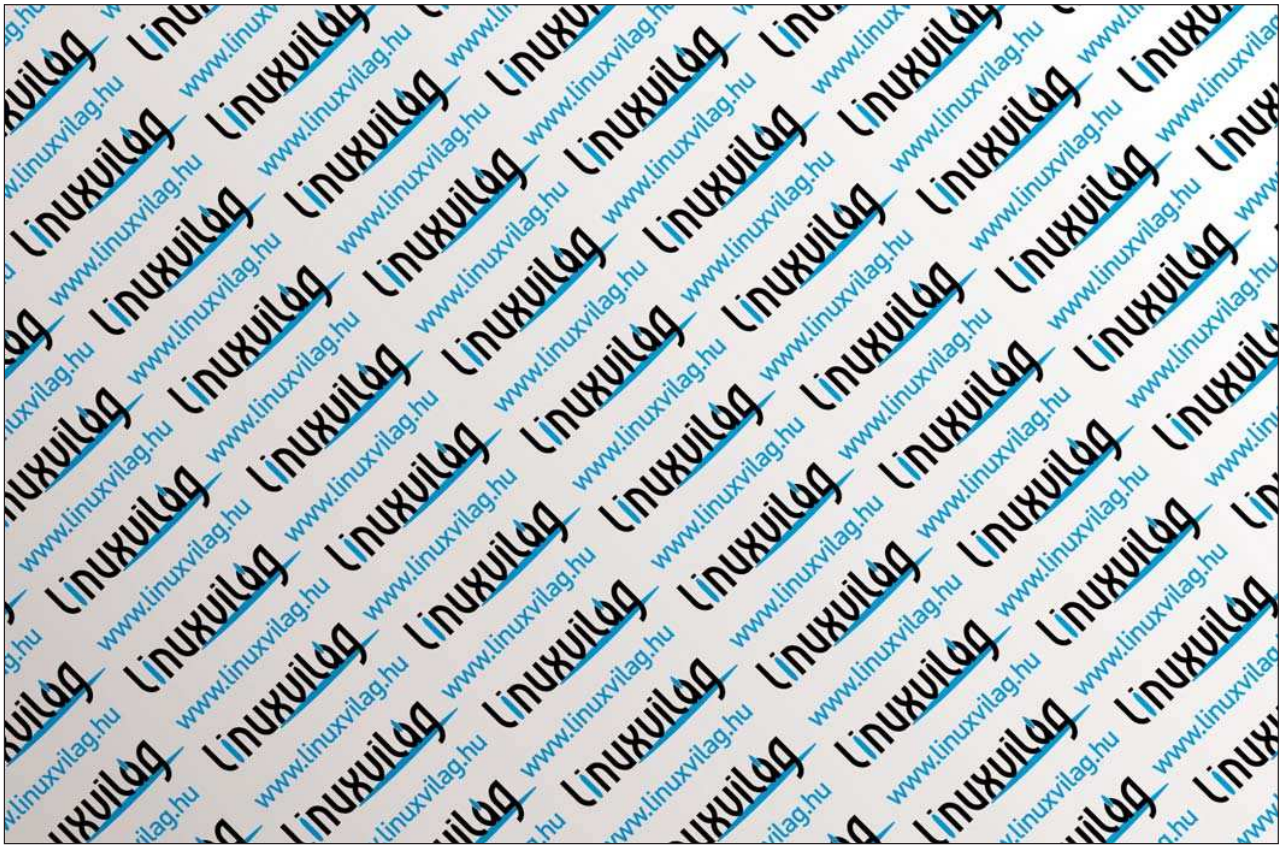
A PostgreSQL lehetővé teszi, hogy függvényeket adjunk meg, amelyeket többféle programnyelv – például szabványos SQL, eljárás alapú Pl/PGSQL, adatbázisokra kihegyezett Perl-változat, Python vagy Tcl – segítségével valósíthatunk meg.
Természetesen a Bricolage adatmodelljének magját a táblák képezik. A ***person*** tábla tartalmazza a rendszer felhasználóit, az ***org*** tábla írja le a szervezeteket, és a ***person_org*** tábla kezeli az előző két tábla metszetét.
A Bricolage belsejének működését könnyen megérthetjük, ha tanulmányozzuk az adatbázis-meghatározásokat. Például egy új cikket – egy áradozó kritikát a Core Perlről – adok hozzá Bricolage rendszerhez az alapértelmezett beállításokkal, azaz az alapbeállításokat használó rendszergazda nevében. A művelet hatására a ***story*** táblába beszúródott egy új rekord, hozzá lett rendelve a fontosság, a megjelenés és a lejárat dátuma, a változatszám (ugyanis a Bricolage a cikkek változatait is kezeli), és annak jelzése, hogy a cikkváltozat megjelent-e.
Számos kulcs – amelyek idegen kulcsok más táblákhoz – jelzi, hogyan illeszkedik bele az adott cikk a rendszerbe. Láthatjuk, hogy a rendszergazda hozta létre, mert az ***usr*** oszlop az ***usr*** táblára mutat; része a ***story*** munkafolyamatnak, ami elkülönül a többi megadott munkafolyamattól, és a ***workflow__id*** oszlopban hivatkozik rá egy külső kulcs. Ez az ***edit*** munkaasztalról érkezett, amely megkülönböztethető a többi munkaasztaltól (például legal és ***publishing***), és az ***edit*** külső kulccsal hivatkozik a ***desk*** táblára; és ez egy ***book review*** (könyvkritika) típusú cikk, mert az ***element__id*** oszlop az ***element*** táblára mutat. Ezek a táblák is össze vannak kötve más táblákkal, így további kiegészítő adatokat (például formátum) is tárolhatunk.
Röviden: a ***story*** tábla áll a Bricolage adatmodelljének központjában, ahogy az logikus is egy olyan tartalomkezelő rendszernél, aminek a tartalom köré kell szerveződnie.
Nagyon ajánljuk a Bricolage programot mindazoknak, akik kezdők a relációs adatbázisok világában, és egy olyan nyílt forrású programot szeretnének tanulmányozni, amely kifinomult módon használja őket.

Összegzés

Ebben a hónapban megismerkedtünk a Bricolage programmal, amely egy nyílt forrású, `mod_perl`-en és PostgreSQL-en alapuló tartalomkezelő rendszer. Megtanultuk a telepítését és a használatba vételét, aztán kicsit beleástuk magunkat az adatmodellbe, amelyet a Bricolage a cikkek elemeinek a nyilvántartására használ. A következő hónapban megnézzük, hogyan kell elemeket, kategóriákat és médiatípusokat megadni, ami lehetővé teszi, hogy ne csak a rendszert piszkáljuk, hanem tényleges tartalmat tehessünk ki a webhelyünkre. Ez után még jobban elmerülünk majd a rendszerben, megvizsgáljuk a Mason-sablonokat, amelyeknek a segítségével olyanra változtatható a Bricolage alapértelmezett kinézete, ami közelebb áll a saját webhelyünkön megjeleníteni kívánt látványtervhez.

Linux Journal 2003. szeptember, 113. 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.



Pillantsunk bele a médiába!

Avagy hogyan használjuk a Linux-rendszert tévénézésre, otthoni videófelvételek készítésére vagy rádióhallgatásra.

Igen, François, ez én vagyok, igaz, hogy húsz évvel ezelőtt, de mégiscsak én. Most meg min mosolyogsz? Azért annyira nem nevetséges a kinézetem. Igen, mondtam már, hogy úgy nézek ki, mintha éppen ***Carl Sagant*** utánoznám, de hát egyszerűen így beszéltem akkoriban, mon ami. Hagyd már abba ezt az idétlen vigyorgást! A vendégeink perceken belül itt lesznek, és a mostani szám központi kérdése – a nyilvános média – olyan téma, amelynek során nagy szükségem lesz a segítségedre a lehető legjobb kiszolgálás érdekében.
Ó, hiszen már itt is vannak! Isten hozott, mes amis! François! A pincébe! A 2000-es Cabernet Sauvignon Stellenbosch Dél-Afrikából most az alkalomhoz illő lesz. Töltsd is ki, kérlek, tout de suite. Kérek mindenkit, hogy helyezze magát kényelembe.
Éppen egy régi videófelvételt mutattam François-nak, amit sok évvel ezelőtt néhány barátommal készítettem. Önkéntesként dolgoztam egy közösségi kábelállomáson, ahol a kamerákat irányítottam és a videóvágóasztalt kezeltem időről időre. Nagyon szórakoztató dolog volt. Egyik nyáron a barátaimmal egy dokumentumfilmet készítettünk a stúdió felszereléseivel, amelynek én voltam az írója és a narrátora. Évekig egy videókazetta formájában hányódott körülöttem az anyag, de az e havi témánk adta ötlet nyomán úgy döntöttem, hogy időállóbb formában tárolom tovább: digitális másolatot készítek belőle. Egy analóg forrásból származó videófelvétel lejátszására egy TV-tuner kártyát használhatunk. Én egy Hauppauge WinTV kártyát szereztem be, ami a btv878-as lapkakészletre épül. A kártya nevével nem nagyon kell törődnünk, mes amis, az újabb Linux-rendszermagok egész jól elboldogulnak vele. Az általam vásárolt kártya egész jól támogatott, a Mandrake 9.1 próbarendszeremen a megfelelő meghajtóprogram önműködően betöltődött (egy másik, Debiannal futó rendszer alatt is szépen működött). A Linux `bttv` rendszermagmodulja vagy meghajtóprogramja nagyszámú tévévevő kártyát támogat. Erről képet kaphatunk, ha egy pillantást vetünk a rendszermag ***CARDLIST*** fájljára (***/usr/src/linux/Documentation/video4linux/bttv/***).
Á, François, jó, hogy visszaértél, tölts, kérlek a vendégeinknek… És fejezd már be ezt az önelégült vigyorgást, nem olyan vicces. Tudjátok, mes amis, hűséges pincérem jól szórakozik azon, ahogy akkoriban, sok évvel ezelőtt kinéztem. Ne is foglalkozzatok vele, inkább élvezzük a bort, s megmutatom, hogyan is működik mindez.
A kártyák elsődleges célja természetesen a tévénézés, de néha FM rádióvevőt is találunk rajtuk. Tévét nézni, miközben dolgozni próbálunk, elég zavaró dolog, rádiót hallgatni már kevésbé problémás. Az egyik FM rádióprogram, ami felkeltette az érdeklődésemet, ***Gerd Knorr*** végtelenül egyszerű alkotása (➲ http://bytesex.org/xawtv). Egy `ncurse` alapú programról van szó, amelyet a `xawtv` forráscsomag részeként kapunk meg. Azért beszélek forráscsomagról, mert ha RPM-csomagokkal dolgozunk, akkor az `xawtv`-től különálló, `radio` nevű csomagban kapjuk a programot.
Gépeljük be a parancssori `radio -s` utasítást. A `-s` kapcsoló használatakor a program kijelzi az állomáskeresés közben vizsgált frekvenciát, majd ezután némítás nélkül indul el. Ha ehelyett a `-i` kapcsolót használjuk, a program a saját könyvtárunkba, egy ***.radio*** nevű fájlba írja a felismert adókat és azok frekvenciáit. Ezután ezt a szöveges fájlt szerkeszthetjük is, adhatunk érdekesebb neveket is az egyes állomásoknak. Hauppauge kártyám nem alkalmas a hang közvetlen lejátszására. Lehetőségem volt arra, hogy a hangszóróimat a kártya kimeneti hangcsatornájára csatlakoztassam, de azt szerettem volna, ha a hangot a rendszeren keresztül tudom vezérelni. Ehhez használnom kellett a tartozékként kapott csatlakozókábelt, amellyel a tévékártya kimenetét a rendszerem hangkártyájához csatlakoztattam. Ezután az ***alsamixer*** segítségével megemeltem a ***line-in*** bemenet szintjét, és már készen is voltam. A kártyánktól függően hasonló módon helyezhetjük üzembe a szerkezetet. Ez a beállítás különösen akkor fontos, amikor a hangrögzítésen kezdünk el gondolkodni.
Egy másik rádióprojekt, ami szót érdemel, ***John Ellis*** GQradio programja (➲ http://gqmpeg.sourceforge.net/radio.html).
A GQradio tulajdonságai közé tartozik az elegáns grafikus felület, az önműködő állomáskeresés, a beállított állomások tárolása és sok egyéb. A honlapon a forráskód mellett megtaláljuk a Red Hat-féle RPM-csomagokat is. A program fordításához szükségünk van a GTK+ és a ***gtk-pixbuf*** programkönyvtárakra. Innentől kezdve a megszokott ötlépéses kicsomagoló–fordító eljárással van dolgunk:

```
tar -xzvf gqradio-0.99.0.tar.gz
cd gqradio-0.99.0
./configure
make
su -c "make install"
```

Ha a programot parancssorból futtatjuk, első indításakor érdekes jelenségre figyelhetünk fel:

```
$ gqradio
Creating dir:/home/mgagne/.gqradio
Creating dir:/home/mgagne/.gqradio/skins
```

A program könyvtárat hoz létre a beállításokat és a bőröket tartalmazó fájlok számára. Tehát a program cserélhető külsővel rendelkezik. Sőt nemcsak bőrök letöltésére nyílik lehetőségünk, a GQradio egy beépített bőrszerkesztő programot is tartalmaz – szabadon engedhetjük alkotói fantáziánkat szárnyalni. Jobb egérkattintás a GQradio grafikus képén, és máris előttünk az a lenyíló menü, amellyel kedvünk szerint módosíthatjuk a program beállításait.
A következőkben a képszórás világa felé fordítjuk figyelmünket, és arra a korai időszakra, amikor a televíziós tudósítások által érintett lettem a média világában. François, talán jobb, ha nekem is töltesz egy pohárkával. Azok után, hogy így kinevettél, jól jöhet egy kis bátorítás.
A türelmetlenek számára elmondom, hogy a legtöbb Linux-

rendszercsomag részeként megkapjuk a `xawtv` tévénéző programot, ami (az `ncurses` alapú rádióprogramhoz hasonlóan) szintén Gerd Knorr munkáját dicséri. Ha egyszerűen elindítjuk a programot, akkor nagy valószínűséggel először csak egy üres képernyővel fogjuk szembetalálni magunkat, főleg, ha Észak-Amerikából próbálkozunk. A program alapértelmezései között szerepel például a PAL képformátum. Ha a földrajzi elhelyezkedésünknek jobban megfelelő beállításokkal szeretnénk körülvenni a `xawtv`-t, a ***$HOME/.xawtv*** beállítófájlt szerkesztenünk kell. Az enyémnek a tartalma a következő:

```
[global]
freqtab = us-bcast

[defaults]
input = Television
norm = NTSC

[vcr]
channel = 3
key = 3
```

A fájl részekre tagolódik, amelyeket egy szögletes zárójelek között lévő cím jelez (a lehetséges beállításokról a `man xawtvrc` paranccsal szerezhetünk ismereteket). Ezek közül a ***global*** (átfogó) és a ***defaults*** (alapértelmezések) című részek a legfontosabbak, mert itt állíthatók be a helyi átviteli szabványok, csakúgy, mint a bemeneti eszközünk. A `[vcr]` szakasz az egyik általam hozzáadott rész, ami egész egyszerűen a 3-as csatornámat, a videomagnóm kimenetét jelenti.
Számos program lehetővé teszi a tévéadások nézését.
Az MPlayer (➲ http://www.MPlayerHQ.hu) egy népszerű, „mindent az egyben” kínáló lejátszóprogram. Erre vonatkozóan is áll a tanács, hogy legelőször a rendszercsomagunkban keressük, valószínűleg van belőle már egy példányunk. A kodekek segítségével az összes közkézen forgó AVI és MPG mozgóképformátum lejátszására képes. Lehet, hogy ezzel nem mondok újat, de az MPlayer tévévevő kártya kezelésére is képes, méghozzá a következőképpen:

```
mplayer -tv
on:driver=v4l:channel=3:input=0:norm=NTSC:widt
h=640:height=480
```

Ha grafikus felületen szeretnénk használni az MPlayert, a `gmplayer` parancsot kell használnunk. A `driver` kapcsoló a video4linux (v4l) meghajtót választja ki. A csatornabeállítás (channel) magától értetődő, a `norm` beállítással pedig kiválaszthatjuk az Észak-Amerikában használatos NTSC sugárzási szabvány használatát. Az utolsó beállítások a kép magasságát és szélességét határozzák meg.
Ezek a programok nagyszerűen használhatók tévénézésre a linuxos rendszerünkön, de ha olyanok vagytok, mint én, akkor kis irigységgel nézitek a kérkedés rossz szokásával bíró barátaitok digitális kábelekkel felszerelt szórakoztató rendszereit.
A TiVo és PVR (Personal Video Recorder) egységeikkel számukra elérhető a következő műsorok képernyőn megjelenő kijelzése, amit így kedvük szerint előre tudnak programozni. Többé nincs ok az irigységre, mes amis. Ha Linux-rendszerünkhöz keresünk egy teljes szórakoztatóközpontot, csak a MythTV programot kell megszereznünk, ezt a minden képességgel felszerelt személyes képmagnó- és televízió-rendszert. Képzeljünk el egy digitális képrögzítő rendszert, amellyel nézhető az

A MythTV felületének beállítása

élő tévéműsor, és lehetőség van az azonnali visszajátszásra, a jelenetek előre-hátratekerésére. A MythTV lehetővé teszi az időzítőn megjelenő műsorok beprogramozását és a hálózaton keresztül elérhető műsorlisták nézegetését, amelyekben karakterláncok keresését is elindíthatjuk – ha éppen a Buffy egy részének az ismétlését keressük. Adjuk még ehhez a több tévévevő kártya kezelésének, és az egyidejűleg több felvétel készítésének a lehetőségét és az osztott rendszert, aminek köszönhetően a hálózaton több különböző MythTV-készülék is beállítható. Vegyük mindezt körül egy tetszetős, témákkal ellátható csomagolással, és így már talán kezd világossá válni, hogy miről is beszélek.
Ez valami olyasminek hangzik, amit azonnal meg kell szereznünk, igaz? Keressük fel a ➲ http://www.mythtv.org címet és töltsük le a szükséges fájlokat. Találunk itt RPM-csomagokat a különböző Linux-változatok számára, csakúgy, mint debianos csomagokat vagy Gentoo ***ebuild*** és ***digest*** fájlokat. További részletekért a ➲ http://www.mythtv.org/docs/mythtv-HOWTO-3.html cím ***Software*** szakaszát érdemes megkeresnünk, ahonnan kiindulva további hivatkozásokat és előre fordított csomagokat találhatunk. Ez a remek programcsomag forráskód formájában szintén a rendelkezésünkre áll.
A MythTV tényleg nagyszerű program, de ahhoz, hogy futtatni tudjuk, szükségünk lesz egy kis előzetes munkára és néhány kellék feltelepítésére. Ezek nagy része fejlesztői programkönyvtár, köztük olyanokkal, mint a freetype2-devel, az XFree86-devel, a qt-devel, a lame és a libexpat. A MythTV azt ajánlja, hogy a ➲ http://sourceforge.net/projects/expat címről töltsük le a libexpat legfrissebb változatát. Végül ahhoz, hogy a MythTV csatornainformáció-letöltő részét és programütemező eszközét használhassuk, szükségünk lesz a Perlre és néhány moduljára. Lehetőségünk van a MythTV infravörös távirányítóval történő vezérlésére is (a lirc segítségével), de ez azok közé a finomságok közé tartozik, amiket már nem próbáltam ki. Előfordulhat, hogy a MythTV működőképessé tételéhez szükségünk lesz további csomagok telepítésére is, ezek között szerepelhet az `XML::Twig`, a `Date::Manip`, az `LWP` és az `XML::Writer`. Telepítésük legegyszerűbb módja a `perl -MCPAN -e` héjparancs használata. Indítsuk el a CPAN-héjat rendszergazdaként, erre meg fog jelenni a `cpan>` parancsjel (prompt). Ha most használjuk először a CPAN-t, túl kell esnünk egy kérdés–válasz részen, ami segít a programnak a helyi CPAN-tükrözések azonosításában. Ezután az alábbi parancsokat kell a `cpan>` parancsjel után begépelnünk:

```
cpan> install XML::Twig
cpan> install Date::Manip
cpan> install LWP
cpan> install XML::Writer
cpan> exit
```

És készen is vagyunk. Az `xmltv` csomag telepítésekor találkozni fogunk egy további telepítésre ajánlott modullal. Jellegzetes Perl-telepítéssel van dolgunk, ami nem áll messze szabályos ötlépéses telepítési eljárásunktól:

```
xmltv-0.5.10.tar.bz2
cd xmltv-0.5.10
perl Makefile.PL
make
su -c "make install"
```

Az XMLTV telepítésének részeként kérdéseket kapunk a területünk listáira vonatkozóan, ebben a szakaszban tehát érdemes figyelni az üzenetekre. Már a célegyenesben vagyunk. Mivel a MythTV a MySQL-t használja az adatok tárolására, egy adatbázist kell számára beállítanunk. Szükségünk lesz a qt-MySQL-csomag betöltésére is (keressük a qt-mysql vagy libqt3-mysql csomagokat). Még ha megfelelően telepített MySQL rendszerrel rendelkezünk is, a csomag nélkül futtatva a ***QMYSQL3 driver not loaded*** szöveghez hasonló hibaüzeneteket kaphatunk. Bizonyosodjunk meg arról, hogy a MySQL telepítve van és fut is, majd használjuk a kapott sémafájlt:

```
cd database
mysql -u root < mc.sql
```

Linux-változatunktól függően előfordulhat, hogy nincs szükség a `-u root` megadására. Még mindig hátravan néhány adatbáziskérdés, amivel foglalkoznunk kell. Pillanatnyilag még csak a helyi (localhost) címről engedélyezett a MythTV adatbázisához való hozzáférés. Én viszont azt szerettem volna, ha a 192.168.22.0 című magánhálózatom minden tagja számára elérhetővé válik az adatbázis:

```
192.168.22.0 private subnet:
$ mysql -u root mythconverg
mysql> grant all on mythconverg.* to
➥mythtv@"192.168.1.%" identified by "mythtv";
```

A fenti százalékjel helyettesítő karakterként szolgál. Ebből következően ha az összes tartomány számára lehetőséget akarunk biztosítani (ezt azért valószínűleg nem szeretnénk), töröljük ki az alhálózati részt, és csak a százalékjelet hagyjuk meg. Néhány lépéstől eltekintve a telepítés meglehetősen egyszerűnek bizonyult. Adódott azért pár kisebb gond, például a Qt fejlesztőcsomag részét képező `qmake` beállító parancsfájlja nem akarta megtalálni a programot, ezért egy közvetett hivatkozást hoztam létre számára a */usr/bin* könyvtárban. Ezután futtattam a `make` parancsot a MythTV könyvtárából, és ezzel meg is oldódott a gond. Most, hogy a programot lefordítottuk és telepítettük, a MythTV könyvtárában találunk egy ***setup*** nevű könyvtárat és benne ugyanezzel a névvel egy futtatható állományt. Futtassuk a következő parancsot: `setup/setup`. Négy beállítási lehetőséget tartalmazó képernyő jelenik meg: ***General*** (általános), ***Capture Cards*** (képrögzítő kártyák), ***Video Sources*** (képforrások) és végül ***Input Connections*** (bemeneti kapcsolatok). Haladjunk végig ezeken, és a helyi rendszerünknek megfelelően állítsuk be a MythTV-t. A bemeneti kapcsolatok (4-es számú) alapjait már a képforrások beállításánál meg kell adnunk. Figyeljünk a képernyőn megjelenő segítő üzenetekre, miközben végighaladunk a folyamaton. Ha minden beállítással készen vagyunk, nyomjuk meg az Esc billentyűt. Most a tévésliták megkeresése céljából futtassuk a `mythfilldatabase` parancsot. Ha ez az első alkalom, hibába ütközhetünk, mert a helyi `xmltv` beállítófájlunk még nem létezik:

```
/usr/bin/tv_grab_na --configure
```

Meg kell adnunk az irányítószámunkat a helyi tévéműsor listákat biztosító szolgáltató azonosításának megkönnyítése érdekében. A lista alapján azt is eldönthetjük, hogy az `xmltv` mely csatornákhoz gyűjtsön műsorinformációkat. Mivel ez az egyik dolog, ami színesebbé teszi az életemet, a legnagyobb választék mellett döntöttem, és az összes csatornát engedélyeztem. Az adatbázisnak műsoradatokkal való feltöltése az egyik olyan feladat, amit minden bizonnyal a `cron`-nal szeretnénk majd megoldani.
A következő lépés a `mythbackend` program elindítása. A démonként való futtatást választottam a `-d` kapcsolóval (ha akarjuk, ezt ***rc.local*** indító parancsfájljainkhoz is hozzáadhatjuk). Végül felhasználóként adjuk ki a `mythfrontend` parancsot a MythTV felhasználói felületének futtatásához. Ezen a ponton a grafikus felületen két lehetőségünk adódunk: a beállítások (setup) és a tévé használata (TV). A kurzorbillentyűk segítségével mozoghatunk a képernyőn. A tévét választva tévénézésre, egy felvétel beállítására vagy egy korábban rögzített műsor megtekintésére nyílik lehetőségünk. Böngészhetjük a műsorfüzetet is a mostani vagy a későbbi műsorok kiválasztásához.
Egy további lehetőség fiatalságunk zavarba ejtő képeinek a rögzítése, ilyen számomra az 1984-es Tall Ships Festivalról szóló beszámoló, a „Romancing the Sail". Válasszuk ki a videómagnónknak megfelelő csatornát, vegyük elő a szalagot,és rögzítsük az emlékeinket.
A MythTV kitűnő programcsomag, tele energiával és ígéretekkel. A megszokott tévénéző és rögzítő oldalán kívül olyan további modulok is elérhetők, amikről most idő hiányában nem tudok beszámolni. Nos, mes amis, az idő később jár, nemsokára zárnunk kell az ajtókat. A mai estétől kezdve még az ilyen késői alkalmakkor sem kell majd attól tartanotok, hogy elszalasztjátok a kedvenc programjaitokat, igaz? A votre santé! ***Bon appétit!***

Linux Journal 2003. szeptember, 113. 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

bttv, radio és a xawtv ➲ http://bytesex.org/xawtv
GQradio ➲ http://gqmpeg.sourceforge.net/radio.html
MPlayer ➲ http://www.MPlayerHQ.hu
MythTV ➲ http://www.mythtv.org
Marcel borlapja ➲ http://www.marcelgagne.com/wine.html

Harc és fantázia

E havi játékainkat a harc és a mágia jegyében választottuk ki.

Mindkét terítékre került program leírását olvasóink kérték. A kimerítő ismertetés két cikknyi helyet tölt meg, pótlandó a múlt havi játékhiányt. E kis felvezetés után csapjunk a CD-k közé!

A Rune

A Loki telepítője a `setup.sh` parancscsal indítható a CD-ről. A telepített

anyag mennyisége 650 MB, ezért cserébe Unreal Tournament-szintű grafikát, kitűnő hangot és lebilincselő cselekményt kapunk.

Történet

Ragnar fiatal viking harcos, korának jellegzetes alakja. Apja felkészíti őt a végső férfiasság próbájára. A faluban, ahol élnek, elég feszült a hangulat, ugyanis éppen egy közeli falut rohantak le a fosztogatók. Nagyon kegyetlenek voltak: a nőket és gyerekeket lemészárolták, és falut porig égették. Conrack, a kíméletlen törzsfőnök Loki (no nem a játékairól elhíresült cég, hanem egy viking isten) követője. Istene azt parancsolta neki, hogy rohanjanak le mindenkit. Loki nevét a félelem halk moraja övezi, ő testesíti meg a gonosz energiát, ami az égből érkezik. Ragnar faluja Odinhoz imádkozik, minden istenek atyjához, és Asgardhoz, a viking menny uralkodójához.
Ragnar megkezdi férfiassági próbáját, a fiatal harcos többször is bizonyítja, hogy férfivá érett. Eleget tesz a kötelező rituálénak, de addig nem lehet belőle igazi harcos, amíg csatában meg nem sebesül. Még le sem törölte magáról a beavatása során ráragadt port, üzenet érkezik egy másik megtámadott, körbezárt faluból. Ragnar falujának harcosai véget akarnak vetni Conrack gonoszságainak. Ragnar, a falu új harcosaként apjával együtt részt vesz a portyán. Ragnar reméli, hogy be tudja bizonyítani a bátorságát, és egy harci sérüléssel be tud lépni a Valhallába. Ám a harc közben egy tengeri csatában Conrack elsüllyeszti a hajójukat, így mindenki a hullámsírba vész, még főhősünk, Ragnar is. A víz alatt azonban Odin feltámasztja Ragnart, és puritán egyszerűséggel közli: „Nos, Ragnar, neked küldetésed van!”

Technológia

A Human Head megvette az Epic Games Unreal Tournament motort és több bővítménnyel látta el, beletették az új csontváz animációs rendszert, az új szemcsehatásrendszert és a kiegészített árnyékrendszert. A Human Head azért készítette ezeket a bővítményeket, hogy kialakítsák a valós elemeken alapuló világokat, a természetes külső környezetet és a minél valósághűbb barlangszerű helyeket.
A kibővített árnyékrendszer pontosan jelenik meg az egyenetlen felületeken, valamint a tárgyak szegélyei körül, ami a játékosok számára lehetővé teszi, hogy a hatást ki-bekapcsolhassák a karaktereiken. A csontváz-animációs rendszer lehetővé teszi az életszerű animációkat, ezáltal a játék karakterei „jobban” reagálnak a sérülésekre, természetesen ez igaz az ellenségre és a környezetre is. A szemcserendszer tartalmazza a tűz-, a robbanás-, a füst-, a por- és felhőhatásokat (effect).
A Rune foglalja magában a teljes egyjátékos módot, természetesen különböző többjátékos szintek is megtalálhatók benne, ezekben a játékosoknak lehetőségük nyílik együttműködni a viking harcosokkal.

Jellegzetességek címszavakban

- Drámai egyjátékos történet, amiben a játékosok számos elemmel találkoznak a sötét mélységektől egészen a magas skandináv hegyekig.
- Többjátékos lehetőség, ami tartalmazza „Deathmatch” és a „Team Deathmatch” elemeket (ebben ki is merül a többjátékos mód).
- 25 pálya található benne.
- 15-féle fegyvert használhatunk, többek között harci fejszét, kétkezes kardot, varázsfegyvereket, dobófejszét és harci kalapácsot.
- A fáklya Rangar hősies utazása során mindenütt az utat mutatja, és ha szükséges, akár fegyverként is használható.
- Ellenfelek bőséges kínálata: vikingek sötét seregétől kezdve a földalatti lényeken, emberevő halakon át a szellemekig és a misztikus szörnyekig.
- Harci rendszer, ami lehetővé teszi a játékosoknak, hogy fegyverrel győzzék le az ellenfeleket.
- Látvány, ami lehetővé teszi a karakterek számára, hogy már az első pillanatban lássák az elérhető fegyvereket.

A játék annak ellenére, hogy hátsó kamera nézetű, első személyű (first person). A játékmenet nem túlságosan bonyolult, szerintem többet ki lehetett volna hozni a viking legendából.
Bár az egyszerű nem feltétlenül könnyű. Három nehézségi szinten próbálhatunk szerencsét. Tizenötféle fegyver áll rendelkezésünkre ahhoz, hogy minden létező ellenséget, szörnyet lemészároljunk. Közben jóllakunk a falakon mászó gyíkokból, és mellékesen agyonverhetjük az ellenséget a saját levágott karjával is. Bár nem éltem, és bevallom, nem is élnék a vikingek korában, a programozóknak

Rendszerkövetelmények

- Linux magváltozat 2.2.X
- Pentium II 3D gyorsítókártyával.
- 64 MB RAM szükséges (128 MB ajánlott).
- Videokártya minimum 640×480 felbontással.
- XFree86 3.3.5 vagy újabb, legalább 16 b/p színmélységgel.
- OSS-megfelelő hangkártya
- Merevlemez 700 MB üres hellyel.

Újdonságok a játékok világából

Postal 2 Linuxon is

Ryan Gordon belekezdett a Postal 2 linuxos átültetésébe (➲ http://icculus.org/%7Eicculus/). A Postal első változatát anno még a Loki készítette, a program erőszakos tartalmával vált híressé-hírhedtté, amivel elérte, hogy bizonyos országokban nem is lehet megvásárolni.

Urban Terror 3.0

Az Urban Terror egy elsősorban hálózatos játékra, csoportküldetésre kihegyezett Q3-bővítmény. Figyelem, a letölthető állomány mérete 322 MB! A Quake 3-hoz készült bővítmény a ➲ http://www.urbanterror.net/ címen érhető el.

Medal of Honor próba1

Végre eljutottunk eddig is! Megjelent az első linuxos ügyfél, annak is a próbaváltozata. Honnan máshonnan is lehetne letölteni, mint a ➲ http://icculus.org/betas/mohaa/mohaa-lnxclient-beta1.tar.bz2 címről!

NWN 1.31

Megjelent az Neverwinter Nights 1.31 változata, amely egyben e havi fő témánk. A ➲ http://nwn.bioware.com/downloads/linuxclient.html címről szerezhető be.
A változáslista a ➲ http://nwn.bioware.com/support/patchdetails131.html oldalon érhető el.

Tedd magad próbára te is!

A Transgaming betatest programba kezdett, a programhoz a ➲ http://www.transgaming.com/showthread.php?news=77 címen lehet csatlakozni. Ne higgye azonban senki, hogy a próbaprogram keretében teljes változatú anyagot fognak kipróbálásra átadni. Jelentkezni azonban mindenképpen érdemes, mivel a CVS-változatnál frissebb és újabb anyagot kaphatunk.

Savage linuxos ügyfél

Az iGames Publishing, valamint az S2 Games kiadja a Savage nevű játék linuxos ügyfélét. A bejelentés a ➲ http://www.gamezone.com/news/07_24_03_03_33PM.htm címen olvasható.
➲ http://www.igames.com/
➲ http://www.s2games.com/

America's Army 1.9-es változat

A program taktikai, katonai játék, amelyet az Amerikai hadsereg is támogat. Előnye, hogy teljesen ingyenes.
A ➲ http://www.americasarmy.com/ címről tölthető le.
A hivatalos bejelentés a ➲ http://icculus.org/news/news.php?id=1616 címen olvasható.

a környezet megalkotása sikerült a legjobban (kivétel, amikor végtelen utakon rohangálunk a barlangrendszerekben), minden azt hiteti el velem, hogy igenis részese lehetek a kornak. Nekem mint viking harcosnak véghez kell vinnem a küldetésemet, ami néha iszonytatóan unalmas, mert nagyon hamar meg lehet unni az állandó kapcsolók utáni futkosást.
Ellenfeleink goblinok, zombik, törpék, csontvázharcosok, sötét elfek, nagyméretű rovarfélék, emberevő halak és jéglakó bestiák. Az ellenfelek nevetségesen egyszerű lények, néha árnyékfutást rendeznek egy-egy tereptárgy mögött, vagy egyszerűen alig mozdulnak ránk. A többjátékos rész nagyon gyengére sikerült, ami azért is furcsa, mert a játék ugye az Unreal Tournament grafikai motorjára épül, tehát legalább az örökségül kapott dolgokat beilleszthették volna.

Fegyverek

Kardok: Viking rövid kard; Római kard; Viking széles kard; Dwarven munkakard; Dwarven harci kard; Balták;
Kézi balta: Balta; Viking balta; Sigurd baltája; Dwarven harci balta.
Kalapácsok: Ütött-kopott buzogány; Koboldcsont bot; Hármas buzogány; Dwarven munkakalapács; Dwarven harci kalapács.

Rune: Halls of Valhalla

A Rune: Halls of Valhalla az ünnepelt PC-s játéknak, a Rune-nek a hivatalos többjátékos kiegészítő csomagja (lásd a fenti elemzést, amiben kifejtettem, hogy mennyire gyenge a Rune többjátékos része). A készítői azt ajánlják a Rune-rajongóknak, hogyha tehetik, próbálják ki magukat a világon elérhető online-játékosokkal. Természetesen a program a játékot egy- és többjátékos módban is lehetővé teszi.
Két új többjátékos bővítmény (Headball és Arena) látott napvilágot, ezekben további pályák találhatók, az újdonságok tömege áraszt el bennünket (állítja a reklámszöveg). Amit valójában kapunk: animált lények, beleértve a három új rémisztő női karaktert, akikkel jártunkban-keltünkben meg tudunk küzdeni. A Headball mód egy csoportos játékon alapuló sport, amiben a játékosok kiszakítják a fákat vagy lefejelik az ellenséget, és pontot szerezhetnek, ha gólt tudnak velük dobni. Ez gyilkosság lesz az Arena módban, ahol a játékosok versenyeznek egymással, s közben váltakozó ellenfelekkel párbajoznak. Hasonlóan az eredeti játékhoz a Rune: Halls of Valhalla is az Epic Game Unreal Tournament bővített játékmotorjára épül, és bámulatos élethűséggel animált, a legmesszebbmenőkig részletezett fegyverekkel, egyedi harci rendszerrel próbál meg bennünket lenyűgözni.

Jellemzők

- Két új játékmód: Headball és Arena, valamint számos új karakter.
- Online többjátékos mód, hozzáféréssel az összes eredeti pályához és karakterhez, ami a Rune-ban található, természetesen az új, bővített tartalommal.
- Eredeti zenei tartalom külön audiosávokon, amelyek kimondottan a Rune: Halls of Valhallához készültek.
- 33 új pálya, amelyekben minden játékmód elérhető, beleértve a rajongók által készített 7, új látvánnyal megáldott pályát.
- Javított internetkód.
- Új harci stílus bővített védelmi lehetőségekkel, ami a játékosokat képessé teszi arra, hogy az eldobott fegyvereket el tudják téríteni. Ezenkívül a pajzsokat is feljavították.

Neverwinter Nights

Ebben a hónapban – többek kérésére – górcső alá veszem a Neverwinter Nights programot.
Be kell vallanom, hogy az „isten”-játékok és a hasonló városépítő, hódító játékok nem tartoznak a kedvenceim közé, de tájékoztatni szeretném a játékos kedvű „nagyérdeműt”. Az természetesen más kérdés, hogy mire mindent leírtam róla, én is rabja lettem a Neverwinter Nightsnak.

Rendszerkövetelmények

A Neverwinter Nights futtatásához szükséges éppen időszerű hivatalos linuxos rendszerkövetelményeket a ➲ http://nwn.bioware.com oldalon nem leltem, ezért a windowsos alapokból indultam ki (lásd a *táblázatunkban*).

Mi is a Neverwinter Nights?

A Neverwinter Nights (NWN) hatalmas, középkori fantáziavilágba ágyazott játék, kazamatákkal és sárkányokkal. Ez a szerepjáték (RPG) egy hősi mesébe repít el bennünket, ahol hit, háború és árulás irányítja életünket.
A játékos, azaz te kiválaszthatod, hogy milyen ügyességi és képességi szinttel rendelkezz – ennek megfelelően tudod felépíteni az utazásodat, úgy, mintha az elfelejtett királyság összetett és veszélyes világában élnél.
Lehetsz veszélyes csavargó, aki az árnyékok leplét használva lop és titkolódzik, vagy tudós és varázsló, aki hatalmas varázslatokat használ ellenségei ellen. Belebújhatsz a behemót barbár bőrébe, aki csatákért epekedik, ahol végre kiélheti szörnyű szenvedélyét. Páncélos Paladinként védelmezheted az ártatlanokat és legyőzheted az ellenségeiket. De lehetsz akár a vitézlő lelkész is, aki meggyógyítja a betegeket és megóvja a gyámoltalanokat… légy mindez, és még több!
A Neverwinter Nights lehetővé teszi, hogy létrehozd a saját világaidat. Ez a forradalmi játék az összes olyan eszközt a rendelkezésedre bocsátja, ami ahhoz szükséges, hogy elkészítsd a saját kalandjaid színhelyét. A Neverwinter Nights Aurora eszközkészlet minden kezdő felhasználó számára lehetővé teszi, hogy mindent elkészítsen: a csendes, misztikus erdőktől az ocsmány, gonosz csöpögő barlangján át a királyi palotáig. Az összes szörny, eszköz részekre szedhető és a világok építői beállíthatják őket. De ne állj meg itt: készíts csapdákat, viadalokat, egyedi szörnyeket és varázslatos elemeket, hogy a játékod, a küldetésed egyéni legyen.
A Neverwinter-élmény nemcsak magányos kaland, játsszál az összes barátoddal! A program elérhetővé teszi, hogy online akár 64 játékossal játssz, akikkel mindent meg tudsz osztani a kaland során. Te irányíthatod és futtathatod a kalandot, és kalandmesterként irányíthatod a szereplőket és vezérelheted az összes szörnyet, lényt és karaktert, akikkel a barátaid az utazásuk során találkozhatnak. Az erőteljes program tartalmazza a Neverwinter Nightsot, a DM-ügyfelet, amivel közelről tudod vezérelni a kalandot, a küldetést mind magad, mind pedig a barátaid számára. A Neverwinter Nights egy végtelen kaland!

A linuxos ügyfél telepítése

1. A fájlt az ➲ ftp://jeuxlinux.com/bioware/Neverwinter_Nights/nwresources129.tar.gz helyről érdemes letölteni (figyelem, a fájl mérete 1,2 GB!).
A fájlt a `tar xvfz-` *`fájl_neve`* paranccsal lehet kicsomagolni, és a mérete ellenére nem tartalmaz mindent, csak annak a három CD-nek az anyagát, amit amúgy is meg tudnál venni.
2. Le kell tölteni az ügyfélprogramot a ➲ http://nwdownloads.bioware.com/neverwinternights/linux/129/nwclient129.tar.gz címről, ezt szintén `tar xvfz-` utasítással kell kicsomagolni. A kicsomagolást követően futtatni kell a `./fixinstall` parancsot (ezt a továbbiakban érdemes megjegyezni, mivel minden frissítést követően le kell futtatni).

A windowsos ügyfél telepítése

Ha a játékot már feltelepítettük egy windowsos lemezrészre, az alábbi könyvtárakra és fájlokra lesz szükségünk:

```
ambient/*
data/*
dmvault/*
hak/*
localvault/*
modules/*
music/*
nwm/*
override/*
portraits/*
saves/*
servervault/*
texturepacks/*
chitin.key
patch.key
dialog.tlk
dialogF.tlk
```

Megjegyzendő, ha a Windows-telepítés csak részleges volt, akkor hozzá kell adni az ***nwn.ini*** fájlhoz az alábbiakat, és be kell fűzni a „Play” feliratú CD-t

Neverwinter Nights rendszerkövetelmények

	Szükséges	Ajánlott
Processzor	Pentium II 450 MHz vagy AMD K6 450 MHz	Pentium III 800 MHz vagy Athlon 800 MHz
RAM	128 MB	256 MB
Merevlemez	1,2 GB	2,0 GB
CD-ROM vagy CD/DVD-ROM-meghajtó	8×	
Videokártya	16 MB TNT2-osztályú OpenGL 1.2-megfelelő videokártya	nVidia GeForce 2/ATI Radeon
Többjátékos mód	IPX vagy TCP/IP LAN-hoz vagy internethez	
Többjátékos mód modemmel	56 k (max. két játékos)	Széles sávú elérés

(természetesen a */mnt/cdrom* változhat annak függvényében, hogy milyen a Linuxunk beállítása):

```
[Alias]
CD0=/mnt/cdrom
AMBIENT=/mnt/cdrom/ambient
MUSIC=/mnt/cdrom/music
```

Telepítés a Loki alapú telepítőprogram használatával

Az elérhető telepítő sajnos nem hibátlan, sok dolgot fel sem rak a windowsos telepítő CD-ről – a CD-mellékletünkre felkerült, de csak erős idegzetűeknek ajánlom kipróbálásra, legalább egyórás bütykölés árán lehet életre kelteni a játékot.

A telepítések végén bármelyiket is választottuk, érdemes a legújabb frissítéseket is felrakni. A jelenlegi az 1.31-es, ez mellékletünkön is megtalálható.
A frissítéshez a `tar xvfz- fájl neve` utasítással csak ki kell csomagolnunk az anyagokat, minden fájlt felül kell írni, ha rákérdez a tarprogram, majd futtatnunk kell a már jól ismert `./fixinstall` utasítást.
A játék futtatása a következő parancsokkal lehetséges: `./nwn`, `:/dmclient` (Dungeon Master client), `./nwserver`, természetesen annak megfelelően, hogy ki miként szeretne játszani: ügyfélként, esetleg hálózatos játékot vagy Neverwinter-kiszolgálót akar-e indítani.

A linuxos kinevezett kiszolgáló telepítése

Amennyiben nem akarunk linuxos ügyfelet futtatni, csak kiszolgálót, a következőket kell tennünk:

1. Telepítsük fel a Neverwinter Nights programot Windows alá.
2. Frissítsük a programot az 1.31-es változatra (még mindig Windows alatt).
3. Másoljuk át a szükséges fájlokat a windowsos lemezrészről a linuxos könyvtárba.

Szükséges fájlok

data/*
override/*
chitin.key
dialog.tlk
patch.key
xp1.key (csak a Shadows of Undrentide küldetéslemezhez szükséges)

Ajánlott fájlok

nwm/*
modules/*
hak/*
nwnplayer.ini

Fontos az ***nwn.ini*** fájlt még véletlenül sem szabad átmásolni, mivel Win32-höz szükséges beállításokat tartalmaz.

4. Csomagoljuk ki a ***linuxserver131.tar.gz*** fájlt a linuxos Neverwinter Nights könyvtárba.

Haladó telepítési mód

Lemezterületet takaríthatunk meg, ha az alábbi fájlokat nem másoljuk fel a linuxos rendszerünkre, mivel szükségtelenek a kinevezett kiszolgáló futtatásához:
data/convo.bif
data/loadscreens.bif
data/music.bif
data/sounds.bif
data/textures_01.bif
data/textures_02.bif
data/voicesets.bif
data/xp1_sounds.bif
data/xp1_textures.bif

Technológia

A játék grafikai motorját a Bioware-Black Isle-Interplay hármas jegyzi.
A fejlesztő a Bioware, egy kanadai csapat, akiket a műfaj kedvelőinek nem kell bemutatni. Legalább ilyen közismertségnek örvend az MDK2 és a Shattered Steel. A NWN az MDK2 Omen motorjának a továbbfejlesztett változatát használja, amely az Aurora névre hallgat. A motor a Bioware saját fejlesztése. A követelményeknek megfelelően a játék háromdimenziós megjelenítést használ. A nézet sokban hasonlít a Baldur's Gate sorozatban már megszokotthoz. Az NWN alapesetben izometrikus megjelenítést használ. A 3D-motor ellenben lehetővé teszi a nagyítást, kicsinyítést, valamint a kamera elforgatását a karakter körül.
A játék három felületre készül: Windows (NT4 és 2000 is), Macintosh és legnagyobb örömünkre Linuxra is. Az NWN vásárlásával nem kapjuk meg mindhárom változatot, bár a küldetéslemezen már rajta van a linuxos anyag, külön telepítővel. A szerkesztőprogram csak Windows alatt fut.
A játék a D&D Elfeledett Birodalmak (Forgotten Realms) világában játszódik, a Faerun nevű földrészen. Az időpont 1372.

A harcrendszer

- ***Valós idejű harc***
  A NWN valós idejű harcot (Real-time Combat) alkalmaz. A harc folyamán a sorrend meglehetősen bonyolult módon kerül meghatározásra – egyaránt befolyásolja a karakterek sebessége, valamit az úgynevezett sebzésmódosítók, ezeket viszont a karakterek fizikai jellemzői, illetve a különféle mágikus hatások határozzák meg.
  A karakternek hat másodpercenként lesz lehetősége rá, hogy valamilyen harci cselekedetet hajtson végre. Ha egyszer kiadtuk az utasítást a karakterünknek, hogy támadjon meg egy adott lényt, akkor ezt addig fogja folytatni, amíg ez az ellenfél meg nem hal vagy újabb parancsot nem kap. A különleges támadási formák, például a célzott lövés, a hárítás, a lefegyverzés vagy a varázslatok nem ismétlődnek önműködően.
- ***AC***
  Az AC (Armor Class – páncélosztály, védettségi fok) azt a nehézségi szintet jelenti, amivel egy ellenfél olyan ütést képes mérni a karakterre, hogy ez az ütése sebzést okozzon. Az AC értéke 10-től 30-ig terjed, s alapvetően a karakter által viselt páncél határozza meg a nagyságát; ezt többféle tényező is módosíthatja, például a DEX-érték, a különféle mágikus hatások vagy a harci módosítók. Minél magasabb ez az érték, annál jobb. Ez az új rendszer váltotta fel a korábbi kiadások THAC rendszerét. A másik (a támadói) oldalt nézve a karakter egy 1d20 kockával dob, a kapott értéket többféle tényezővel módosítja: az STR-értékkel, mágikus hatásokkal, karakterosztálytól függő tényezőkkel, szintfüggő tényezőkkel stb., s így kapjuk meg azt az AC-értéket, amit az adott karakter még meg tud ütni.
  A fegyveres harc a legegyszerűbb formájában abból áll, hogy a karakter támadóértéke összevetésre kerül az ellenfele AC-értékével.
  A sebzés mértékét már a fegyver típusa és különféle módosítók adják meg. A támadás kritikus ütést is eredményezhet, ami jóval nagyobb sebzést okozhat. Ha a karakter a támadódobása meghatározásakor 20-as értéket dob, újradobhat. Ha a második dobásnál is olyan értéket dob, ami ütést eredményez, kritikus ütésről beszélünk.

Ekkor a sebzés meghatározásához kétszer dobják ki a sebzés értéket, s a két számot összegzik.

- ***Mentődobások***
  A mentődobások (Saving throws) adják meg a karakternek azt a képességét, hogyan képes ellenállni bizonyos hatásoknak, illetve varázslatoknak. Ezek az értékek a karakter szintjének növekedésével fokozatosan javulnak. A sikeres mentő azt jelenti, hogy a karaktert egyáltalán nem, vagy csak csökkentett mértékben érinti az adott hatás.
  Az új rendszer szerint a mentőknek három fő csoportjuk van: gyorsaság (Reflex), akaraterő (Will) és állóképesség. Mindháromnak van egy alapértéke, amelyet befolyásolnak az alaptulajdonságok, a karakterosztály-függő, illetve a szinthez kapcsolódó módosítók. A DEX a Reflex, a WIS a Will, a CON a Fortitude mentődobásokra van hatással.
- ***Varázslás (Spell Casting)***
  A papok (cleric), druidák (druid), varázslók (wizard), mágusok (sorcerer) és a bárdok (bard) képesek varázsolni. A meghatározott varázslatszintekhez kapcsolódó varázslatok számát a karakterosztály és a karakter szintje határozza meg. Bónuszvarázslatokat kaphatnak a papok és druidák a magas WIS-érték, a varázslók a magas INT, a mágusok és bárdok a magas CHAR-érték miatt.
  A varázslók rendelkeznek még néhány olyan bűbájjal is, amelyeket naponta megtanulhatnak. Ezeket a bűbájokat trükköknek is szokták nevezni, és igazából nulladik szintű varázslatnak felelnek meg. A papoknak és druidáknak most már nem kell imádkozni az isteneikhez a gyógyító varázslatokért, viszont a varázslataikat átalakíthatják gyógyító energiává.
  Ha egy varázshasználó sérülést szenved egy adott körben, még mindig képes varázsolni, ha sikeres koncentrációpróbát hajt végre.
  Az ellenfeleknek további három lehetőségük van, hogy megszakítsák a varázslat befejezését:
- ***Készenlét (Ready)***: ez egy olyan cselekedet, ami lehetővé teszi a harcban résztvevőnek, hogy meghatározzon egy bizonyos eseményt – ennek a bekövetkeztekor fog cselekedni.
- ***Ellenvarázslat (Counterspell):*** az a cselekedet, amikor a varázshasználó megpróbálja meghiúsítani egy másik varázshasználó varázslatát, azáltal, hogy ugyanazt a varázsigét ugyanabban az időben mondja el.
- ***Megszakítás (Disrupt):*** a mágiatörés (Dispel Magic) egyedi alkalmazása.

Neverwinter Nights: Shadows of Undrentide

A történet szerint Hilltop falujában él egy Drogan nevű törpe, aki valamikor kalandor volt. Mostanában négy, tehetséges tanítványnak kívánja átadni a tudását. A tanítványok egyike leszünk, a nyugodt hétköznapokat egyszer csak megzavarja valami, a koboldok lerohanják a falut, és elrabolnak négy szent tárgyat, és megmérgezik a mesterünket.
Öt új „prestige” osztályt áll rendelkezésünkre: Arcane Archer, Assassin, Blackguard, Harper Scout, Shadowdancer.

1. Az Arcane Archer harcászati tulajdonságokkal, mágikus tudással bír. Barbárral, harcossal, rangerrel és paladinnal válhatunk Arcane Archerré, s ahhoz, hogy felvehessük ezt a kasztot, bizonyos feltételeknek meg kell felelnünk, mint ahogyan a másik négy kaszt esetében is.
2. Az Assassin gonosz forma, kém, aki bele tud olvadni a környezetébe. Gyakran bérgyilkosként alkalmazzák, mivel a halálos varázslatok mestere. Csak gonosz kaszttal vehetjük fel, ami mellett minimum 8-as Hide és Move Silently skillekkel kell rendelkeznünk.
3. A Blackguard szintén gonosz figura. A kaszt felvételéhez megint csak gonosznak kell lennünk, valamint a Base Attack bónuszunknak minimum +6-osnak kell lennie, Hide skillünknek pedig legalább +5-ösnek.
4. A Harper Scout gyökeresen más, mint az előbbi két kaszt, a jellemük akármi lehet, gonosz kivételesen nem. A kaszt képviselői tagjai egy titkos társaságnak, ami a társadalom és a természet közti egyensúlyt hivatott szabályozni. A legtöbb Harper bárd, ranger, tolvaj, varázsló, illetve mágus. A lopakodás és a kémkedés nagymesterei. A felvételhez szükségesek: az Alertness, az Iron Will feat ismerése, valamint a minimum 4-es Search, a 8-as Persuade, a 6-os Lore és a 4-es Discipline skill értékkel való rendelkezés.
5. A Shadowdancer a Shadows of Undrentide által nyújtott új és misztikus karakter, aki világosság és sötétség határán áll, nagymestere az átveréseknek. Tulajdonsága lehet bármi, jó, rossz is, és bármelyik kaszttal felvehetjük. Ismernünk kell a Dodge és Mobility feateket, valamint legalább 8-as Move Silently, 10-es Hide és 5-ös Tumble skill értékkel kell rendelkeznünk.
   A többjátékos bővítmény 3 új építőelemmel lett bővítve. A játékban lényeges újítás nincs, tulajdonképpen csak a Shadows of Undrentidebe beépített dolgokat tudjuk felhasználni az alap NWN-ben is. A Shadows of Undrentide nagyon rövid, csak akkor érdemes megvenni, ha a modulkészítések miatt van szükségük az új építőelemekre.

Telepítés

A CD-n rajta van a linuxos anyag is, így azt nem kell, sőt jelen pillanatban nem is lehet letölteni. A telepítéshez a következőket kell tenni:

```
cd ~/nwn
unzip
/mnt/cdrom/Data_Shared.zip
unzip
/mnt/cdrom/Language_data.zip
unzip
/mnt/cdrom/Language_update.zip
unzip
➥/mnt/cdrom/Data_Linux.zip
rm data/patch.bif
rm patch.key
./fixinstall
```

Megjegyzés: amennyiben az unzip rákérdez, minden egyes fájlt felül kell írni.
Ezután érdemes felrakni a frissítést, ami szintén helyet kapott mellék-

KAPCSOLÓDÓ CÍMEK

Magyar oldalak

NWN-magyarítás
➲ http://www.napnet.hu/stevesum/

Hivatalos oldalak

A hivatalos Neverwinter Nights oldal
➲ http://nwn.bioware.com/

A Bioware honlapja
➲ http://www.bioware.com/

D&D 3. kiadásának hivatalos honlapja
➲ http://www.wizards.com/3e/

NWN online-világok

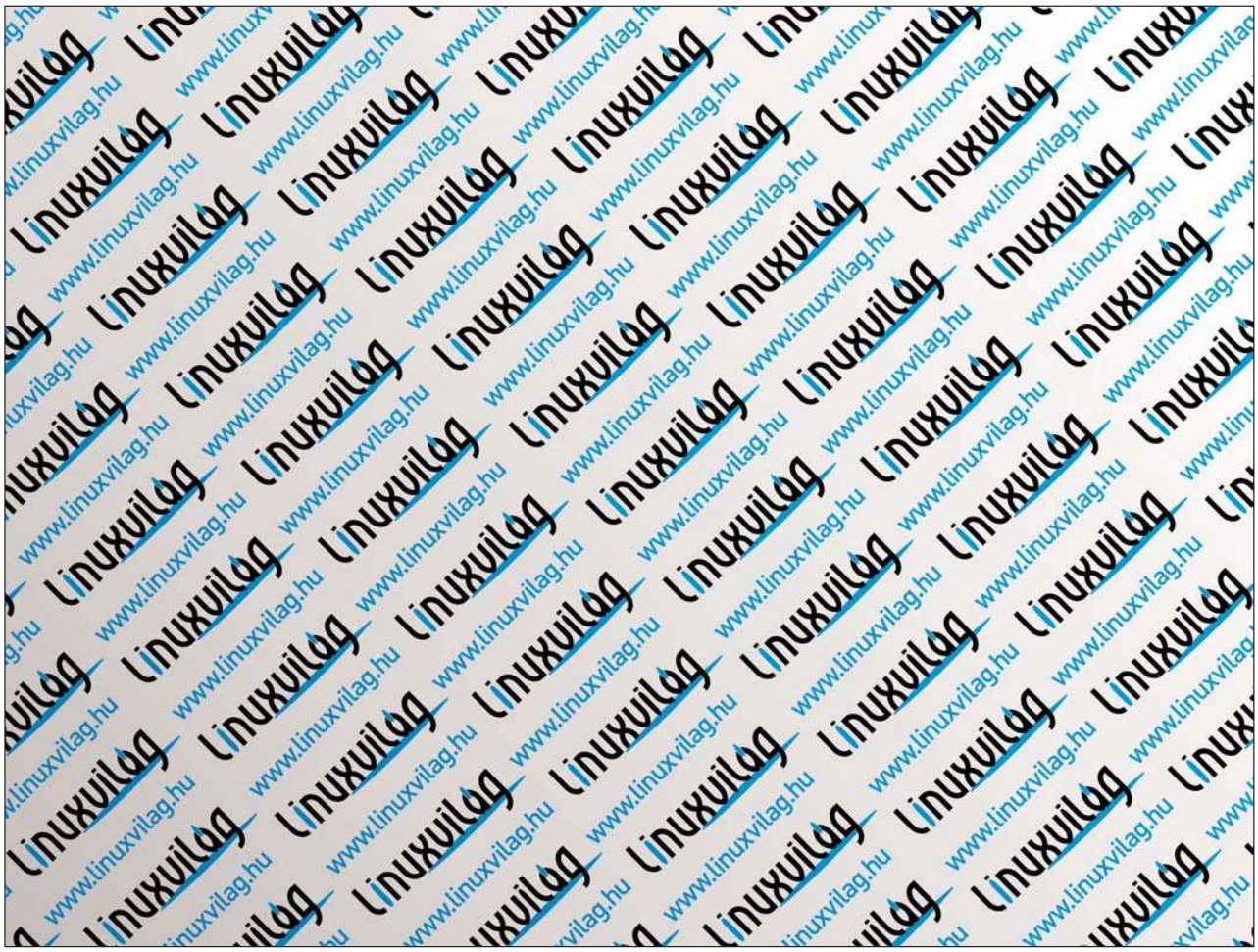
ALFA project (24 órás online világ)
➲ http://www.alandfaraway.net/

A Gemworlds Neverwinter Nexus készíti
➲ http://www.neverwinternexus.net/gemworlds/

letünkön (***linuxsouclientupddate*** néven kell keresni). A már megszokott módon járjunk el: `tar xvfz- fájl neve`, majd `./fixinstall`.
A korongra felkerült még az ***nwn.zip*** fájl, ami tartalmazza az ***nwn.ico***, ***nwn.xpm*** és NWN inifájlt. Ezenkívül a jó regisztrációs kulccsal elérhető oldalakat is feltettük HTML-változatban. A CD-re felkerült még az NWN végigjátszása magyarul és a SOU Hilltop végigjátszása eredetiben. Említésre méltó még a magyar oldal (➲ http://www.neverwinternights.hu), ahonnan rengeteg leírás, anyag tölthető le. Többek között a nyelvi fájl is (***dialog.tlk***), amely a letöltéskor még csak 1.30-as változatnál tartott. Fontos, hogyha a fordított változatot ki szeretnénk próbálni, mindenképpen készítsünk egy másolatot az eredeti 1.31-es angol ***dialog.tlk*** fájlról, mivel a fordítás, gondolom, nem végleges állapotú, így hol magyar, hol angol a szöveg. Ami még meglepőbb volt, az az, hogy a SOU használata során sokkal inkább érvényesültek a fordítások, mint az eredeti NWN-nel való játékkor, pedig mind a két küldetés 1.31-esre lett javítva. Utoljára még egy jó tanács: én mind Red Hat 9, mind pedig Xandros 1.1 alatt kipróbáltam a játékot. Míg a Red Hat 9 esetében semmilyen gond nem volt, addig a Xandros alatt a kicsomagolt tarfájl (nwnresources) rossz tulajdonossal hozta létre a könyvárakat és fájlokat. Erre csak akkor jöttem rá, amikor esze ágában sem volt elfogadni a CD-kulcsot. Miért is? Mert nem tudta létrehozni a kulcsot tároló fájlt. Ugyanilyen gond volt, hogy Xandors alatt az összes futtatandó fájl jogosultságainál be kellett kapcsolni a setuid bitet, mert a játék csak és kizárólag rendszergazdai jogosultsággal akart futni.
Minden nehézség ellenére állítom, hogy a játékra érdemes több időt szánni, és elmélyedniük benne még azoknak is, akik nem szeretik a szerepjátékokat vagy a fantáziavilágokat, mivel az én példámból is okulva, még érhetik meglepetések. A végén csak azon veszi észre magát az ember, hogy nincs külvilág, nincs semmi, csak a város, a szörnyek és a varázslatok, amelyekben elmerülve várunk az újabb és újabb kalandra.

Köszönetnyilvánítás
Köszönetet kell mondanom többeknek is, elsősorban ***Székely Leventének***, közvetlen főnökömnek, aki támogatta és támogatja munkámat, valamint a cégnek, ahol dolgozom, a Diamond Electric Kft.-nek, mivel a próbagép árát a vállalat fizette. Külön öröm, hogy korábbi kérésemre a Xandrostól is most érkezett válasz, így a gépen a Xandros Desktop 1.1 Deluxe kereskedelmi változata fut, amelyet teljesen ingyen bocsátottak a rendelkezésemre.

Kosztadinovszki Norbert
(kosztadinovszki@diaelec.hu)
Linux- és játékmániás számítógépőrült.



Növeljük az X hatékonyságát!

Fogjuk munkára az X-et parancsfájlokkal, testreszabott billentyűzettel és párbeszédablakokkal!

A Linux rendelkezik grafikus felhasználói felülettel (GUI), amelyen több ablaklezelő és menürendszer is található. De a sebességet tekintve a grafikus felület szűk keresztmetszetet jelent az ember és a számítógép kapcsolatában: a gép sebességétől függetlenül ugyanannyi időt vesz igénybe egy olyan folyamat, amelyik tíz egérkattintást igényel tíz különböző helyen. A parancssorban a hagyományos Unix válasza a szerszámosláda-szemlélet. Mindent gépesíthetünk parancsfájlokkal, és apró, egy adott feladatot végrehajtó programokat csővezetékkel köthetünk össze. A harmadik megoldás ezt a két megközelítést egyesíti, és elég egyszerű és hatékony ahhoz, hogy a legtöbb célra megfeleljen. Ablakkezelőtől függetlenül bármilyen X-környezet a szokásosnál gyorsabbá és rugalmasabbá tehető.

Saját egér és billentyűzet

Több ujjunk van, mint kezünk, és 101 billentyű jut egy egérre. Bár a rajzolás és az ehhez hasonló feladatok gyorsabban elvégezhetők egérrel, az ujjainknak eközben is a billentyűzeten kell lenniük. Ha egy parancs vagy választási lehetőség egyetlen művelettel elvégezhető, egy-két billentyű lenyomása sokkal gyorsabb az egérkattintásnál, nem beszélve az ínhüvelygyulladás veszélyeiről. Az is jó lenne, hogy amikor az egérhez érünk, azonnal azt tegye, amire szükségünk van.
Az `xmodmap` segítségével a billentyűk és az egérgombok kiosztása megváltoztatható, így különleges jelentéssel ruházhatók fel (például ékezetes betűk), vagy GUI-műveleteket rendelhetünk hozzájuk (például az ablak maximalizálását). Az egyik előny, hogy a balkezes emberek a két egérgombot megcserélhetik. Adjuk hozzá a következő sort a ***.xinitrc*** állományhoz, vagy tegyük az idézőjeles részt a ***.xmodmaprc*** állományba:

```
xmodmap -e "pointer = 3 2 1"
```

Az utóbbi időben az `xmodmap` már kezelni tudja az egérgörgőket Linux alatt (lásd a ***Kapcsolódó címek*** részt). Gyorsabbá is tehetjük a görgőkkel való munkát, ha a leggyakoribb műveleteket az újfajta egerek különleges gombjaihoz rendeljük. Például az IntelliMouse Explorerhez ez a bűvös sor szükséges:

```
xmodmap -e "pointer = 1 2 3 6 7 4 5"
```

Amikor a billentyűzetről beszélünk, a módosító olyan billentyűt jelent, ami megváltoztatja más billentyűk állapotát vagy hatását. Szabványos módosító a SHIFT, CTRL, LOCK, ALT és öt további, amelyeket egyszerűen MODN-nek (N=1,2,..5) hívnak. Az `xmodmap` tetszőleges billentyűhöz rendelheti hozzá őket. A leggyakoribb talán a súgóoldalon is leírt eset, amikor a CTRL és a CAPS LOCK billentyűt cserélik fel. Ugyanígy a Windows billentyű mod4-re változtatható – feltéve, hogy a billentyűkódja 115:

```
xmodmap -e ´keycode 115 = Alt_R Meta_R´
```

Grafikus előtétprogram a szolgáltató választásához

```
#! /bin/bash

/bin/rm -f /tmp/netsettings

echo -n "PROVIDER=" > /tmp/netsettings
Xdialog --menubox "Szolgáltató választása"
                  ➥20 40 5
"ISP_1"  "Munkaidőben a legolcsóbb"
"ISP_2" "Hétvégén gazdaságosabb"
"ISP_3"      "Leggyorsabb FTP" 2>>
             ➥/tmp/netsettings

RETVAL=$?
# add control code here

echo -n "ACCOUNT=" >> /tmp/netsettings
Xdialog --inputbox "Írd be a fiók nevét"
➥10 25 "son" 2>> /tmp/netsettings

echo -n "LOGFILE=" >> /tmp/netsettings
Xdialog -fselect /tmp 20 80  2>>
➥/tmp/netsettings

source /tmp/netsettings
netconn $PROVIDER $ACCOUNT $LOGFILE
```

A billentyűk alkalmazások indítására is programozhatók, de ez már függ az ablakkezelőtől. A Blackbox 0.65-ben például a `bbkeys` segédprogrammal lehet billentyűleütéseket és alkalmazásokat egymáshoz rendelni. A ***$HOME/.bbkeysrc*** állományban a

```
KeyToGrab(F1), WithModifier(None),
➥WithAction(ExecCommand), DoThis(xterm
➥-geometry 80x25 -e mutt)
```

sor azt eredményezi, hogy az F1 billentyű lenyomására egy 80×25 méretű `xterm` ablakban elindul a `mutt`. Számos más ablakkezelő is lehetővé teszi az ilyen hozzárendeléseket – olvassuk el a leírásokat.

Kattintgatós parancsfájlok

Minden parancsfájl használhat ablakokat a felhasználóval való párbeszéd során, és minden grafikus ügyfél (maga az X is) veheti a bemenetét közvetlenül egy szöveges programból. Az első eset akkor következhet be, ha a parancsfájlnak a felhasználó visszajelzésére van szüksége, de ez a hagyományos módon nem lehetséges; például a felhasználó inkább meghal, csak ne kelljen gépelni, vagy nincs billentyűzet (utcai termi-

nálok). A második esetre példa, amikor a felhasználónak előre nem kiszámítható szöveget kell egy konzolos programból egy olyan X-es programnak átadnia, amelybe a szöveg kézzel nem másolható át. Gyakran előfordul ez a mindennapi életben: a `mutt` és a `lynx` elegendő lehet a levelek olvasására és a webböngészésre, de mi van, ha valamelyik levél vagy weboldal azt mondja: „Nézd meg ezt a filmelőzetest”, és ekkor a Mozillát indítanánk el egy egérkattintással…?

Parancsfájlok bővítése ablakokkal

Legalább tíz éve már X-ablakokat is lehet parancsfájlokból nyitni. Kezdetben ott volt a ma már nem működő Xscript. A korszerű megoldás az Xdialog, a GTK+ alapú párbeszédablak-építő. A *listánkban* közölt parancsfájl akár a valós életből is származhatna: a felhasználó kiválaszthatja a legjobb internetszolgáltatót, a használandó fiókot és a kapcsolat naplójának a helyét. Ezután elindul egy hagyományos ***pppd/chat*** parancsfájl (itt netconn a neve), ami átveszi a grafikus felületről begyűjtött adatokat. Vizsgáljuk meg részletesen a grafikus felületet!

1. kép Az Xdialog indítása

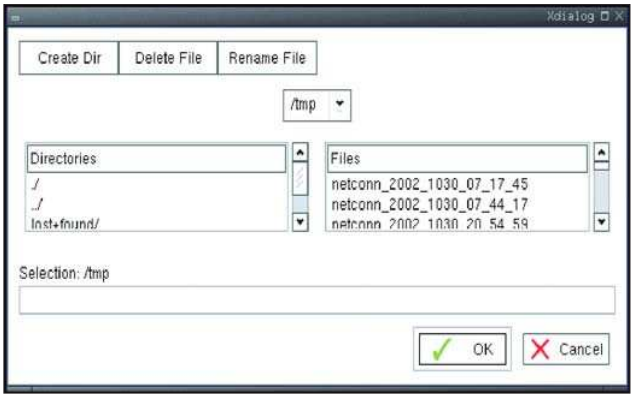
A parancsfájl azzal indul, hogy a felhasználó régi választásait tartalmazó ***netsettings*** állományt eltávolítja. A `netconn` parancsfájl által igényelt összes változó a bash parancsértelmező értékadási szabályai szerint a */tmp/netsettings* állományba kerül. A bal oldali rész egyszerűen soremelés nélkül kiíródik (`echo -n`). A jobb oldali részt, azaz a felhasználó által választott értéket az Xdialog adja át.

2. kép A fióknév bevitele

Az első párbeszédablakban (*1. kép*) a felhasználó kiválaszthatja a legjobb internetszolgáltatót. Megadjuk a párbeszédablak leírását (szolgáltató választása), az ablak magasságát (20) és szélességét (40), valamint a beviteli mezők magasságát (5). Az 1. képen bemutatott esetben, miután a felhasználó megnyomta az ***OK*** gombot, a */tmp/netsettings* a következő sort fogja tartalmazni:

```
PROVIDER=ISP_1
```

Az Xdialog minden meghívása után a kilépési kód vizsgálatával (a RETVAL az 1. listában) érdemes hibaellenőrzést végeznünk. Így ellenőrizhetjük, hogy a felhasználó mit is tett valójában. A rövidség kedvéért ezt a kódrészt most elhagytuk, de közöljük, hogy a `RETVAL 1` értéke azt jelenti, hogy a „Mégsem gombot nyomták meg”, a `255` azt jelenti, hogy a felhasználó bezárta az ablakot, és a `0` jelenti azt, hogy a választás megtörtént. A második Xdialog parancs lehetővé teszi, hogy a felhasználó begépelje a fiók nevét vagy elfogadja az alapértelmezett beállítást (*2. kép*). Az utolsó párbeszédablak (*3. kép*) egy fájlválasztó ablakot jelenít meg, amelynek segítségével a felhasználó kiválaszthatja a naplófájl helyét és nevét. Ezután a felhasználó választásai már a */tmp/netsettings* állományban vannak, ezért ezt az állományt csak be kell tölteni (source), és el kell indítani a kapcsolatot.



3. kép A naplófájl kiválasztása

Az Xdialog nagyszámú más építőelemet is tartalmaz, például választógombokat, csúszkákat és naptárat. Ezek közül kevés használható a felhasználó számára történő visszajelzésre. A szerző a `--msgbox` kapcsolót használja felbukkanó ablak megjelenítésére, amelyben például fel lehet sorolni, hogy fiókok szerint rendezve hány üzenetet töltött le a `fetchmail`. Vannak még folyamatjelzők, adott idő után eltűnő információs ablakok és fájlmegjelenítő ablakok. Az Xdialog használható CD-égető, hang- és videolejátszó, valamint biztonsági mentést készítő parancsfájlok létrehozására – a felhasználási lehetőségeknek csak a képzeletünk szabhat határt.

Mi történik akkor, ha a parancsfájlt akkor kell használni, amikor nem fut az X? Semmi gond, karakteres terminálokon is meg lehet menüket és ablakokat jeleníteni. A `dialog` programot kell használnunk, ami az Xdialog karakteres megfelelője.

Szöveg átadása az X-nek

Az URL-es példára visszatérve: gépelés nélkül is át lehet adni egy címet a böngészőnek, ha az `xclip` programot használjuk, amely az egérrel kijelölt szöveget írja ki. Ezt a programot a `gnome-moz-remote` segédeszközzel együtt használjuk, ami vagy egy új Mozilla-példányt indít el, vagy a már futó Mozilla új ablakában megjeleníti a megadott címet. Írjuk be a következő sort egy parancsfájlba:

```
gnome-moz-remote --newwin "´xclip -o´" &>
➥/dev/null &
```

Nevezzük el a parancsfájlt `start_browser.sh`-nak, és rendeljük hozzá egy makrógombhoz egy külső alkalmazásokat indítani képes programban. A `mutt`-ban egy ilyen makró például így néz ki:

```
macro pager \cn "!start_browser.sh\n"
➥´open URL´
```

Ezután, ha a `mutt`-ban egy címet látunk, kijelölhetjük az egérrel és megnyomhatjuk a CTRL-N billentyűkombinációt. A parancsfájl belsejében az `xclip` kiírja a kijelölt szöveget, és minden úgy zajlik, mintha kézzel indítottunk volna egy

böngészőt, és a címsorba ezt a szöveget írtuk volna be.
Az `xclip`-nek rokona az `xclipboard` és az `xcutsel`.
Ez a két program akkor hasznos, ha meg szeretnénk tekinteni a vágólap tartalmát, illetve amikor a kijelölést két olyan alkalmazás között kell átvinni, amelyek ezt nem támogatják.
További tudnivalók a megfelelő súgóoldalakon találhatók.

A kattintás utánzása

Hogyan hitethetjük el az X-szel, hogy az egérrel mozgatjuk a mutatót, és kattintunk a gombokkal, miközben csak a billentyűzetet használjuk? Erre való az `xbut` program. Egy egyszerű állomány segítségével beállíthatjuk, hogy bizonyos billentyűk mozgassák az egérmutatót és utánozzák a kattintást.
Még többre képes az `xwit`, amellyel közvetlenül egy adott pontra helyezhetjük át az egérmutatót, emellett az ablakok áthelyezése és minimalizálása is lehetséges. Az éppen használt `xterm` minimalizálásához, majd két másodperc után az eredeti méret visszaállításához a következő parancsot használjuk:

```
xwit -iconify; sleep 2; xwit -pop
```

Ez a megoldás a sokáig futó feladatok elvégeztetésekor hasznos. A `sleep` helyére írjuk be a sokáig futó parancsot – az ablak eltűnik, majd a munka végeztével újra megjelenik.
Végül bemutatjuk a sokoldalú `xte` eszközt, ami az `xautomation` csomag része. Az `xte -h` parancs kiírja a támogatott X-eseményeket. Az alábbi példa áthelyezi az egérmutatót a képernyő bal szélétől számított 320 képpontra, a felső szélétől számított 50 képpontra levő helyre, és kattint egyet az egyes egérgombbal:

```
xte ´mousemove 320 50´ ´mousedown 1´
➥´mouseup 1´
```

A `mousedown` és a `mouseup` események különállóak, így lehet húzni is, nemcsak kattintani.

Jelszókezelő munkaasztal

Egy gyors és hatékony környezetben a felhasználó minden számára engedélyezett folyamatot a lehető legkönnyebben, leggyorsabban és legbiztonságosabban elindíthat (akár a háttérben, akár interaktívan, akár szöveges, akár grafikus programot), minden olyan gépen, amelyhez hozzáférése van. Természetesen a leghelyesebb mindezt az OpenSSH-n használatával megtenni. De mégha RSA és DSA alapú hitelesítést végzünk is – megtakarítva a jelszavak gépenként történő megjegyzését –, minden kapcsolathoz be kell írnunk az SSH-jelszót. Szerencsére az `ssh-agent` képes megjegyezni a titkos kulcsot vagy kulcsokat, és minden hitelesítési feladatot elvégez, amelyhez a titkos kulcs kell. A gyakorlatban ez azt jelenti, hogy ha az X-et az `ssh-agent` gyermekfolyamataként indítjuk, az `ssh-agent` programot minden később indított helyi X-ügyfél használhatja a hitelesítésre.
Az ablakkezelőt az `ssh-agent` gyermekeként az ***.xinitrc*** vagy az ***.Xsession*** állományban úgy indíthatjuk, hogy az `ssh-agent` után beírjuk az ablakkezelő nevét. Ha `sawfish`-t használunk, írjunk „ssh-agent sawfish”-t, és minden, ami az X-munkamenet alatt fut, beleértve az SSH programokat is, használni fogja az `ssh-agent`-et.
A Gnome az `ssh-agent` programot alapból elindítja. A KDE-hez úgy adhatjuk hozzá, hogy megkeressük a KDE indítóállományát, és úgy járunk el, mint egy ablakkezelő esetében.
Az `ssh-add` parancs az, amelyik ténylegesen megismerteti a kulcsokat az ügynökkel. Ellenőrizhetjük, hogy az `ssh-agent` fut-e, és ismeri-e a kulcsunkat:

```
ssh-add -L
```

Ha a számítógépet őrizetlenül hagyjuk, bárki, aki arra jár, kérdés nélkül azonnal hozzáfér mindenhez, amihez mi is. Jelentkezzünk ki, vagy az `ssh-add -D` paranccsal töröljük a kulcsunkat.

Események azonosítása

Elmondtuk, hogyan lehet egyes billentyűk billentyűkódjait más esemény előidézésére átkódolni, de honnan tudhatjuk meg ezeket a kódokat? A megoldás az xev diagnosztikai eszköz, ami egy eseménytesztelő ablakot nyit meg, és minden ablakeseményt kiír ebbe. A szerző gépén a bal oldali Windows-billentyű lenyomása ezt adja (a „keycode” értéke érdekel minket):

```
KeyRelease event, serial 23, synthetic NO,
➥window 0x1000001, root 0x46, subw 0x0, time
➥1108438536, (175,176), root:(627,425), state
➥0x40, keycode 115 (keysym 0xffeb, Super_L),
➥same_screen YES, XLookupString gives 0
➥characters: ""
```

Végül, de nem utolsó sorban el szeretnénk dicsekedni a grafikus felülettel rendelkező parancsfájlunkkal, és ehhez képernyőképek kellenek. Ebben az esetben is elég az X és az ImageMagick, nem kell ennél csicsásabb előtétprogram. A cikkben használt képek mind a következő néhány szabványos paranccsal készültek, amelyekhez a megfelelő leírást a súgóoldalakon lelhetjük fel:

```
xwd -out temp_image -frame
xwdtopnm temp_image  > fig1.pnm
convert fig1.pnm fig1.png
```

Az első parancs a mutatóval kiválasztott ablakról a kerettel együtt mentést készít. A második és harmadik parancs először PNM, azután PNG formátumra alakítja át a képet. Mondanom sem kell, hogy mindhárom parancs könnyen beilleszthető parancsfájlba, ami megkérdezi a felhasználót, hogy miről készüljön a mentés (képernyő vagy ablak), és hová mentse az eredményt.

Összegzés

Számos felhasználó érzi úgy, hogy egy teljes munkaasztali környezet vagy túl sok mindent tartalmaz, ami lelassítja a számítógépet, vagy túl keveset a különleges igények kielégítéséhez. A cikkben leírt eszközök és módszerek segíthetnek ezeknek a felhasználóknak a termelékenység növelésében, és olyan rendszeren is nélkülözhetetlenek, amelyet felváltva használnak a parancssor és a kattintgatás szerelmesei.

Linux Journal 2003. szeptember, 113. szám

Marco Fioretti
Áramkörtervező mérnök, a szabad programok érdeklik a gépesített áramkörtervezés (EDA) és a hatékony munkaasztal kialakítása (jelenleg a RULE projektet vezeti) szempontjából is. Marco a családjával Rómában él.